前　言

MATLAB 是英文 MATrix LABoratory 的缩写，意为矩阵实验室，其产生的最初目的是为软件中的矩阵运算提供方便。MATLAB 是一种解释性语言，它采用技术计算语言，几乎与专业领域中所使用的数学表达式相同。不同工具箱中的专业函数相对独立。MATLAB 中的基本数据元素是矩阵，它提供了各种矩阵的运算和操作，并有较强的绘图能力，所以得以广为流传，并成为当今国际控制界应用最广、备受人们喜爱的一种软件环境。除了在控制界应用外，MATLAB 还在生物医学工程、信号分析、语言处理、图像信号处理、雷达工程、统计分析、计算机技术和数学等各行各业中都有极其广泛的应用。

虽然 MATLAB 最初并不是为控制理论与系统的设计者们编写的，但是 MATLAB 软件一出现就很快引起了控制界研究人员的瞩目，因为它把看起来相当繁琐复杂的矩阵操作变得简单得令人难以置信，加上 MATLAB 还能十分容易地绘制各种精美的图形，从而吸引了世界上控制界的许多名家，在自己擅长的领域编写了许多具有特殊意义的 MATLAB 工具箱，这又反过来使得 MATLAB 更加具有吸引力。因为利用 MATLAB 及其工具箱所得结果是相当令人信服的，所有数字及其数据处理以及专业公式的计算都是基于公认的算法，由国际上各有关的顶尖专家编写而成。至今为止，MATLAB 环境下的与控制界有关的工具箱有：

控制系统工具箱（Control system）
频域系统工具箱（Frequency Domain system）
鲁棒控制工具箱（Robust control）
系统辨识工具箱（System identification）
信号处理工具箱（Signal processing）
最优化工具箱（Optimization）
神经网络工具箱（Neural network）
样条工具箱（Splines）
模糊逻辑工具箱（Fuzzy logic）
小波工具箱（Wavelet）
μ 分析与综合工具箱（μ-analysis and synthesis）
非线性控制设计工具箱（Nonlinear control design）

MATLAB 之所以有如此强大的功能是在于其还在不断扩大的工具箱的应用上，离开了工具箱的应用，MATLAB 环境下的操作也仅仅是简单的矩阵运算与作图而已。而要想利用工具箱为己服务，必须知道工具箱能做什么？它的理论依据是什么？它能解决什么问题？达到什么目的？可以这么说，每一个工具箱都有一门专业理论作为背景的。它是为这个理

论服务的。它将该理论中所涉及到的公式运算、方程求解包括复杂的矩阵操作，全都编写成了 MATLAB 环境下的子程序。设计者只要根据自己的需要，通过直接调用函数名，输入变量，运行函数，便可立即得到结果，从而大大节省了设计人员的编程和运算求解的时间。因而 MATLAB 工具箱具有很强的专门知识要求，它是为设计人员在运用某一专门理论解决问题时所提供的有效快捷的工具。只有在掌握了其理论的基础上才能够明白工具箱中的每一个函数的意义，所要达到的目的和所要解决的问题，才能够正确地使用它们，使工具箱很好地为自己服务。

神经网络工具箱正是 MATLAB 环境下所开发出来的许多工具箱之一。它是以人工神经网络理论为基础，用 MATLAB 语言构造出典型神经网络的激活函数，如： S 型、线性、竞争层、饱和线性等激活函数。使设计者对所选定网络输出的计算，变成对激活函数的调用。另外，根据各种典型的修正网络权值的规则，加上网络的训练过程，用 MATLAB 编写出各种网络权值训练的子程序。网络的设计者则可以根据自己的需要去调用工具箱中有关神经网络的设计与训练的程序，使自己能够从繁琐的编程中解脱出来，集中精力去思考问题和解决问题，从而提高效率和解题质量。

正是因为 MATLAB 环境下的工具箱的使用需要较强的专业理论知识，所以本书的重点放在对人工神经网络理论的介绍以及与神经网络工具箱应用的结合点上。已经学会使用 MATLAB 语言的读者，通过本书可以掌握人工神经网络的理论，并能够较快的学会用其工具箱设计出网络来解决实际问题。已掌握人工神经网络理论的读者，通过本书可以学会用 MATLAB 设计、训练网络，使理论上的神经网络开始为解决实际问题服务，从理论走向实际应用。这一步，如果没有 MATLAB 的帮助，实现起来将是较为困难的。即使对于已有过实际应用的读者，通过本书也可以从中了解到许多很难发现或证实的神经网络特性与功能，能够对神经网络有更进一步的理解与认识，为其应用开拓思路，同时再加上学会了运用 MATLAB 环境下的神经网络工具箱，将使得设计者如虎添翼，有能力设计出功能更强、更有效的神经网络。

为此，本书在每一章里都给出了大量的应用实例，并全部采用 MATLAB 以及神经网络工具箱中的函数来求解。从而使读者能够采用工具箱中的函数直接设计训练网络，快速、准确地看到神经网络解决问题的能力以及各种网络的特性。这使得读者通过本书的学习，在理解和掌握了人工神经网络理论的同时，又可以运用 MATLAB 程序来设计网络，将其用于解决实际问题。即使对于目前还没有 MATLAB 及其工具箱的读者，也能够通过本书的学习，掌握人工神经网络的各类模型的设计与应用。由于书中有各种图形以及性能的对比，使读者对每种网络的优缺点有更加透彻地了解。在掌握了工作原理之后，读者完全可以自己动手采用其他语言编写出解决自己问题的软件来。不过这要比采用 MATLAB 工具箱的编程费时得多。

在顺序安排上，本书在每一章中首先介绍网络构造、基本原理、学习规则以及训练过程。然后通过实例、图解分析或比较网络的优点及其局限性，并提出解决问题的方法。通过和读者一起做练习和观察运行 MATLAB 程序后的解答，从中了解和掌握人工神经网络的特点与 MATLAB 环境下神经网络工具箱的妙用。

本书在人工神经网络理论方面重点放在以下典型网络模型的结构特性以及功能的介绍与应用上：感知器（Perceptron）、自适应线性元件（Adaptive Linear Element）、反向传播

网络（Back Propagation Networks)、霍普菲尔德网络（Hopfield Networks)、内星、外星和科荷伦学习规则（Instar, Outstar, Kohonen Learning Rules）、自组织竞争网络（Self - organization Competition Networks)、特性图（Kohonen Feature Maps)、对传网络（Counter Propagation Networks)、自适应共振理论（Adaptive Resonauce Theory)。

关于 MATLAB 环境下的基本操作、语句结构以及运算，读者可以参考有关 MATLAB 书籍及本书所列出的参考文献。另一个最简单的办法，是通过 MATLAB 环境下的 help 命令在线自学。在这里，我们只是对所用到的神经网络工具箱中的函数以及有关的 MATLAB 编程语言的用法在适当的时候作详细具体的解释。

本书是笔者在中国科学技术大学教授了三年高年级本科生《人工神经网络》选修课讲义的基础上，充实整理后完成的。本书可作为计算机、电子学、信息科学、通讯、控制等专业的高年级本科生、研究生以及其他专业科技人员学习神经网络或学习 MATLAB 及其神经网络工具箱时的教材或参考书。由于笔者水平有限，不当之处在所难免，敬请读者批评指教。

从 爽

1998 年 8 月

于中国科学技术大学

目　次

第一章 概 述

1.1 人工神经网络概念的提出

人脑是宇宙中已知最复杂、最完善和最有效的信息处理系统，是生物进化的最高产物，是人类智能、思维和情绪等高级精神活动的物质基础，也是人类认识较少的领域之一。长期以来，人们不断地通过神经学、生物学、心理学、认知学、数字、电子学和计算机科学等一系列学科，对神经网络进行分析和研究，企图揭示人脑的工作机理，了解神经系统进行信息处理的本质，并通过对人脑结构及其信息处理方式的研究，利用大脑神经网络的一些特性，设计出具有类似大脑某些功能的智能系统来处理各种信息，解决不同问题。

用机器代替人脑的部分劳动是当今科学技术发展的重要标志。计算机就是采用电子元件的组合来完成人脑的某些记忆、计算和判断功能的系统。现代计算机中每个电子元件的计算速度为纳秒（10^{-9}秒）级，而人脑中每个神经细胞的反应时间只有毫秒（10^{-3}秒）级。然而在进行诸如记忆回溯、语言理解、直觉推理、图像识别等决策过程中，人脑往往只需要一秒钟左右的时间就可以完成复杂的处理。换句话说，脑神经细胞做出决定需要的运算不超过 100 步，范德曼（J.A.Feldman）称之为 100 步程序长度。显然，任何现代串行计算机决不可能在 100 步运算中完成类似上述的一些任务。由此人们希望去追求一种新型的信号处理系统，它既有超越人的计算能力，又有类似于人的识别、判断、联想和决策的能力。

人工神经网络（Artificial Neural Network, 简称 ANN）正是在人类对其大脑神经网络认识理解的基础上人工构造的能够实现某种功能的神经网络。它是理论化的人脑神经网络的数学模型，是基于模仿大脑神经网络结构和功能而建立的一种信息处理系统。它实际上是由大量简单元件相互连接而成的复杂网络，具有高度的非线性，能够进行复杂的逻辑操作和非线性关系实现的系统。

人工神经网络吸取了生物神经网络的许多优点，因而有其固有的特点:

（1）高度的并行性

人工神经网络是由许多相同的简单处理单元并联组合而成，虽然每个单元的功能简单，但大量简单处理单元的并行活动，使其对信息的处理能力与效果惊人。

（2）高度的非线性全局作用

人工神经网络每个神经元接受大量其它神经元的输人，并通过并行网络产生输出，影响其他神经元。网络之间的这种互相制约和互相影响，实现了从输人状态到输出状态空间的非线性映射。从全局的观点来看，网络整体性能不是网络局部性能的简单迭加，而表现出某种集体性的行为。

（3）良好的容错性与联想记忆功能

人工神经网络通过自身的网络结构能够实现对信息的记忆。而所记忆的信息是存储在神经元之间的权值中。从单个权值中看不出所储存的信息内容，因而是分布式的存储方式。

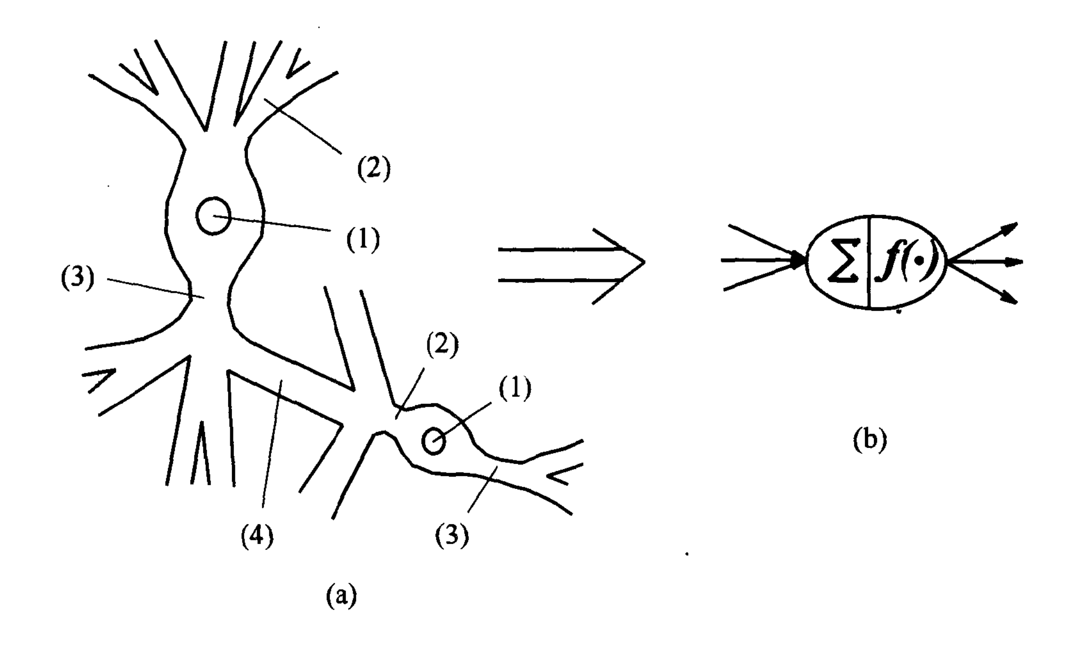
这使得网络具有良好的容错性，并能进行聚类分析、特征提取、缺损模式复原等模式信息处理工作；又宜于做模式分类、模式联想等模式识别工作。

（4）十分强的自适应、自学习功能

人工神经网络可以通过训练和学习来获得网络的权值与结构，呈现出很强的自学习能力和对环境的自适应能力。

1.2 神经细胞以及人工神经元的组成

神经系统的基本构造单元是神经细胞，也称神经元。它和人体中其他细胞的关键区别在于具有产生、处理和传递信号的功能。每个神经元都包括三个主要部分：细胞体、树突和轴突。树突的作用是向四方收集由其他神经细胞传来的信息，轴突的功能是传出从细胞体送来的信息。每个神经细胞所产生和传递的基本信息是兴奋或抑制。在两个神经细胞之间的相互接触点称为突触。简单神经元网络及其简化结构如图 1.1 所示。



(a)简单神经元网络图　　　　(b) 简化后的网络示意图

(1) — 细胞体; (2) — 树突; (3) — 轴突; (4) — 突触

图 1.1 简单神经元网络及其简化结构图

从信息的传递过程来看，一个神经细胞的树突，在突触处从其他神经细胞接受信号。这些信号可能是兴奋性的，也可能是抑制性的。所有树突接受到的信号都传到细胞体进行综合处理。如果在一个时间间隔内，某一细胞接受到的兴奋性信号量足够大，以致于使该细胞被激活，而产生一个脉冲信号。这个信号将沿着该细胞的轴突传送出去，并通过突触传给其他神经细胞。神经细胞通过突触的联接形成神经网络。

人们正是通过对人脑神经系统的初步认识，尝试构造出人工神经元以组成人工神经网络系统来对人的智能，甚至是思维行为进行研究；尝试从理性角度阐明大脑的高级机能。经过几十年的努力与发展，已涌现出上百种人工神经网络模型。它们的网络结构、性能、算法及应用领域各异，但均是根据生物学事实衍生出来的．由于其基本处理单元是对生物神经元的近似仿真，因而被称之为人工神经元。它用于仿效生物神经细胞最基本的特性，与生物原型相对应。人工神经元的主要结构单元是信号的输入、综合处理和输出，其输出信号的强度大小反映了该单元对相邻单元影响的强弱。人工神经元之间通过互相联接形成网络，称为人工神经网络。神经元之间相互联接的方式称为联接模式，相互之间的联接度由联接权值体现。在人工神经网络中，改变信息处理过程及其能力，就是修改网络权值的过程。

目前多数人工神经网络的构造大体上都采用如下的一些原则:

1）由一定数量的基本单元分层联接构成;

2）每个单元的输入、输出信号以及综合处理内容都比较简单;

3）网络的学习和知识存储体现在各单元之间的联接强度上。

1.3 人工神经网络应用领域

随着人工神经网络技术的发展，其用途日益广泛，应用领域也在不断拓展，已在各工程领域中得到广泛的应用。总而言之，人工神经网络技术可用于如下信息处理工作：函数逼近、感知觉模拟、多目标跟踪、联想记忆及数据恢复等。具体而言，其主要用于（或比较适宜于用来）解算下述几类问题:

（1）模式信息处理和模式识别

所谓模式，从广义上说，就是事物的某种特性类属，如：图像、文字、语言、符号等感知形象信息；雷达、声纳信号、地球物探、卫星云图等时空信息；动植物种类形态、产品等级、化学结构等类别差异信息等等。模式信息处理就是对模式信息进行特征提取、聚类分析、边缘检测、信号增强、噪声抑制、数据压缩以及各种变换等。模式识别就是将所研究客体的特性类属映射成“类别号”，以实现对客体特定类别的识别。人工神经网络特别适宜解算这类问题，形成了新的模式信息处理技术。它在各领域中的广泛应用是神经网络技术发展的重要侧面。这方面的主要应用有：图形、符号、手写体及语音识别，雷达及声纳等目标识别，药物构效关系等化学模式信息辨识，机器人视觉、听觉，各种最近相邻模式聚类及识别分类等等。

（2）最优化问题计算

人工神经网络的大部分模型是非线性动态系统，若将所计算问题的目标函数与网络某种能量函数对应起来，网络动态向能量函数极小值方向移动的过程则可视作优化问题的解算过程。网络的动态过程就是优化问题计算过程，稳态点则是优化问题的局部或全局最优动态过程解。这方面的应用包括组合优化、条件约束优化等一类求解问题,如任务分配、货物调度、路径选择、组合编码、排序、系统规划、交通管理以及图论中各类问题的解算等。

（3）信息的智能化处理

神经网络适宜于处理具有残缺结构和含有错误成分的模式，能够在信源信息含糊、不确定、不完整，存在矛盾及假象等复杂环境中处理模式。网络所具有的自学习能力使得传统专家系统技术应用最为困难的知识获取工作转换为网络的变结构调节过程，从而大大方便了知识库中知识的记忆和抽提。在许多复杂问题中（如医学诊断），存在大量特例和反例，信息来源既不完整，又含有假象，且经常遇到不确定性信息，决策规则往往相互矛盾，有时无条理可循，这给传统专家系统应用造成极大困难，甚至在某些领域无法应用，而神经网络技术则能突破这一障碍，且能对不完整信息进行补全。根据已学会的知识和处理问题的经验对复杂问题作出合理的判断决策，给出较满意的解答，或对未来过程作出有效的预测和估计。这方面的主要应用是：自然语言处理、市场分析、预测估值、系统诊断、事故检查、密码破译、语言翻译、逻辑推理、知识表达、智能机器人、模糊评判等。

（4）复杂控制

神经网络在诸如机器人运动控制等复杂控制问题方面有独到之处。较之传统数字计算机的离散控制方式，更适宜于组成快速实时自适应控制系统。这方面的主要应用是：多变量自适应控制、变结构优化控制、并行分布控制、智能及鲁棒控制等。

（5）信号处理

神经网络的自学习和自适应能力使其成为对各类信号进行多用途加工处理的一种天然工具，尤其在处理连续时序模拟信号方面有很自然的适应性。这方面的主要应用有：自适应滤波、时序预测、谱估计和快速傅里叶变换、通信编码和解码、信号增强降噪、噪声相消、信号特征检测等。神经网络在做弱信号检测、通信、自适应滤波等方面的应用尤其引人注目，已在许多行业得到运用。

1.4 人工神经网络发展的回顾

人工神经网络的实质反映了输入转化成输出的一种数学表达式，这种数学关系是由网络的结构确定的，而网络结构必须根据具体问题进行设计和训练。学习人工神经网络的关键在于掌握生物神经网络与人工神经网络建模的联系，人工神经网络的数学基础，以及人工神经网络的应用。下面首先从人工神经网络发展的过程入手来了解和建立生物神经网络与人工神经网络建模的联系，并通过网络结构与数学表达式将其关系联系起来。

一般认为，最早用数学模型对神经系统中的神经元进行理论建模的是美国心理学家麦卡洛克（W. McCulloch）和数学家皮茨（W. Pitts）。1943 年，他们在分析和研究了人脑细胞神经元后认为：人脑细胞神经元的活动像一个断通的开关。为此他们引入了阶跃阈值函数，并用电路构成了简单的神经网络模型，这就是常称的M P模型，如图 1.2 所示。图中输入矢量分量为p_j ($j = 1, 2, ..., r$)；神经元输出状态取值为1或0，权值分量为w_j ($j = 1, 2, ..., r$)，体现了输入矢量作用的强弱：其值为+1 时，表示该输入节点处于兴奋状态，产生加强的作用；其值为-1 时，则起抑制作用。函数$f(\cdot)$被称为激活传递函数（Activation transfer function），常简称为激活函数或传递函数。激活函数的输入为神经元输入矢量的加权和：

$\sum_{j=1}^{r} w_j p_j$ ，其输出代表神经元的输出。所以激活函数反映了人工神经元的实质内容。在MP神经元的数学模型中，模仿断通开关功能的激活函数是一个二值型阀值函数，其数学表达式为:

$$a = f(n) = \begin{cases} 1 & \sum_{j=1}^{r} w_j p_j \geq 0 \\ 0 & \sum_{j=1}^{r} w_j p_j < 0 \end{cases} \tag{1.1}$$

其输入/输出关系式如图 1.3 所示。

MP 神经元模型首次用简单的数学模型模仿出生物神经元的活动功能，并揭示了通过神经元的相互连接和简单的数学计算，可以进行相当复杂的逻辑运算这一令人兴奋的事实。代表 MP 神经元模型的阀值型激活函数在以后的许多其他人工神经网络中得到了广泛的应用。

1957 年，美国计算机学家罗森布拉特（ F. Rosenblatt ）提出了著名的感知器（ Perceptron ）模型。它是一个具有连续可调权值矢量的 MP 神经网络模型，经过训练可以达到对一定的输入矢量模式进行分类和识别的目的。

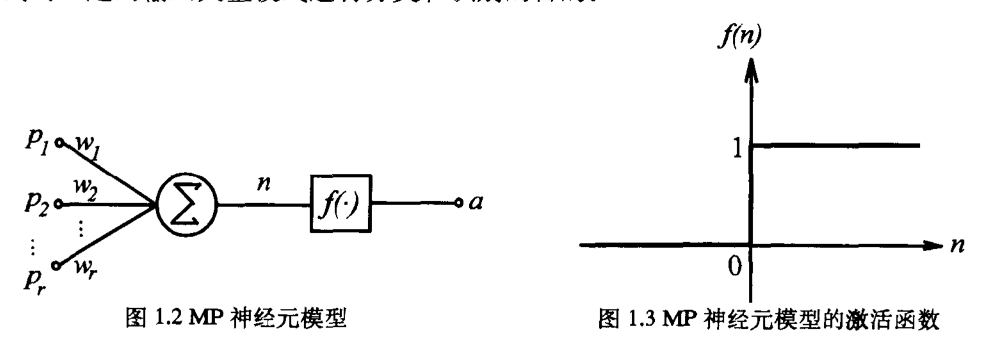


图 1.2 MP 神经元模型　　　　图 1.3 MP 神经元模型的激活函数

1959 年，当时的另外两位美国工程师威德罗（ B. Widrow ）和霍夫（ M. Hoff ）提出了自适应线性元件(Adaptive linear element, 简称 Adaline)。它是感知器的变化形式，尤其在修正权矢量的算法上进行了改进，不仅提高了训练收敛速度，而且提高了训练精度。他们从工程实际出发，不仅在计算机上模拟了这种神经网络，而且还做成了硬件。并将训练后的人工神经网络成功地用于抵消通讯中的回波和噪声，成为第一个用于解决实际问题的人工神经网络。

1969 年，人工智能创始人之一明斯基（ M. Minsky ）和帕伯特（ S. Papert ）在合著的《感知器》一书中对以单层感知器为代表的简单人工神经网络的功能及其局限性从数学上进行了深入的分析。他们指出：单层感知器只能进行线性分类，对线性不可分的输入模式，哪怕是简单的“异或”逻辑运算，单层感知器也无能为力，而其解决办法则是设计训练出具有隐含层的多层神经网络。同时他们还指出，在引入隐含层后，要找到一个有效地修正权矢量的学习算法并不容易。这一结论使得当时许多神经网络研究者感到前途渺茫，客观上对神经网络理论的发展起了一定的消极作用。

美国学者霍普菲尔德（J. Hopfield）对人工神经网络研究的复苏起到了关键性的作用。1982年，他提出了霍普菲尔德网络模型，将能量函数引入到对称反馈网络中，使网络的稳定性有了明确的判据，并利用所提出的网络的神经计算能力来解决条件优化问题。另外，霍普菲尔德网络模型可以用电子模拟线路来实现，这种神经网络所执行的运算在本质上不同于布尔代数运算，从而由此它还兴起了对新一代电子神经计算机的研究。

另一个突破性的研究成果是儒默哈特（D. E. Rumelhart）等人在1986年提出的解决多层神经网络权值修正的算法—误差反向传播法（Error Back-Propagation），简称BP算法，找到了解决明斯基和帕伯特提出的问题的办法，从而给人工神经网络增添了活力，使其得以全面迅速地恢复发展起来。

1.5 人工神经网络的基本结构与模型

从上节对人工神经网络发展的回顾中我们可以了解到人工神经网络是在现代神经学、生物学、心理学等科学研究成果的基础上产生的，反映了生物神经系统的基本特征，是对生物神经系统的某种抽象、简化与模拟。具体人工神经网络是由许多并行互联的相同神经元模型组成，网络的信号处理由神经元之间的相互作用来实现。

一个人工神经网络的神经元模型和结构描述了一个网络如何将它的输入矢量转化为输出矢量的过程。这个转化过程从数学角度来看就是一个计算的过程。也就是说，人工神经网络的实质体现了网络输入和其输出之间的一种函数关系。通过选取不同的模型结构和激活函数，可以形成各种不同的人工神经网络，得到不同的输入/输出关系式，并达到不同的设计目的，完成不同的任务。所以在利用人工神经网络解决实际应用问题之前，必须首先掌握人工神经网络的模型结构及其特性以及对其输出矢量的计算。

1.5.1 人工神经元的模型

神经元是人工神经网络的基本处理单元，它一般是一个多输入/单输出的非线性元件。神经元输出除受输入信号的影响外，同时也受到神经元内部其它因素的影响，所以在人工神经元的建模中，常常还加有一个额外输入信号，称为偏差(bais),有时也称为阀值或门限值。

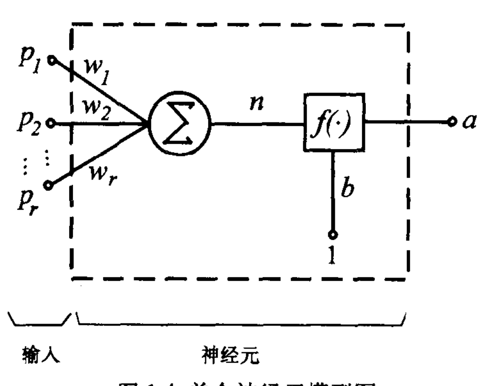


图 1.4 单个神经元模型图

一个具有 r 个输入分量的神经元如图1.4所示。其中，输入分量 $p_j(j=1, 2, ..., r)$ 通过与和它相乘的权值分量 $w_j(j=1, 2, ..., r)$ 相连，以 $\sum_{j=1}^{r} w_j p_j$ 的形式求和后，形成激活函数 $f(\cdot)$ 的输入。激活函数的另一个输入是神经元的偏差 b。

权值 w_j 和输入 p_j 的矩阵形式可以由 W 的行矢量以及 P 的列矢量来表示:

$$W=[w_1\ w_2\ \dots w_r]$$

$$P=[p_1 p_2 \dots p_r]^T$$

神经元模型的输出矢量可表示为:

$$A=f(W*P+b)=f(\sum_{j=1}^{r} w_j p_j+b) \tag{1.2}$$

可以看出偏差被简单地加在 $W*P$ 上作为激活函数的另一个输入分量。实际上偏差也是一个权值，只是它具有固定常数为 1 的输入。在网络的设计中，偏差起着重要的作用，它使得激活函数的图形可以左右移动而增加了解决问题的可能性。

1.5.2 激活转移函数

激活函数 (Activation transfer function) 是一个神经元及网络的核心。网络解决问题的能力与功效除了与网络结构有关，在很大程度上取决于网络所采用的激活函数。

激活函数的基本作用是:

1 ）控制输入对输出的激活作用;

2 ）对输入、输出进行函数转换;

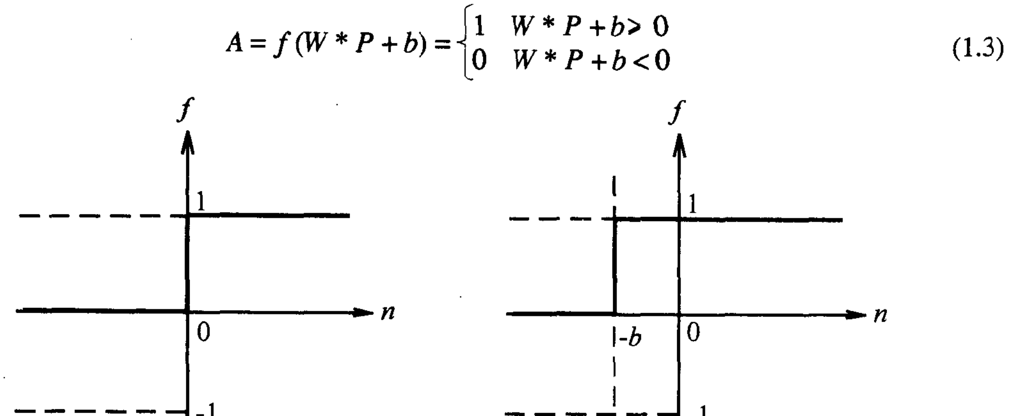
3 ）将可能无限域的输入变换成指定的有限范围内的输出。

下面是几种常用的激活函数。

（1）阀值型（硬限制型）

这种激活函数将任意输入转化为 0 或 1 的输出，函数 $f(\cdot)$ 为单位阶跃函数，如图 1.5 所示。具有此函数的神经元的输入/输出关系为:

$$A=f(W*P+b)=\begin{cases}1 & W*P+b>0\\0 & W*P+b<0\end{cases} \tag{1.3}$$



(a) 没有偏差的阀值型激活函数　　(b) 带有偏差的阀值型激活函数

图 1.5 阀值型激活函数

（2）线性型

线性激活函数使网络的输出等于加权输入和加上偏差，如图 1.6 所示。此函数的输入/输出关系为:

$$A = f(W^*P+b) = W^*P + b \tag{1.4}$$

（3） S 型(Sigmoid)

S 型激活函数将任意输入值压缩到（0,1）的范围内，如图 1.7 所示。此种激活函数常用对数或双曲正切等一类 S 形状的曲线来表示，如对数 S 型激活函数关系为:

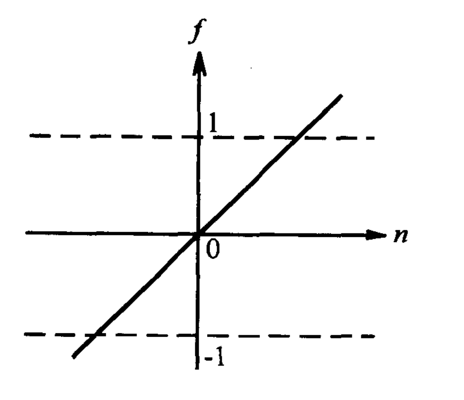
$$f = \frac{1}{1+\exp[-(n+b)]} \tag{1.5}$$

而双曲正切 S 型曲线的输入/输出函数关系为:

$$f = \frac{1-\exp[-2(n+b)]}{1+\exp[-2(n+b)]} \tag{1.6}$$

S 型激活函数具有非线性放大增益，对任意输入的增益等于在输入/输出曲线中该输入点处的曲线斜率值。当输入由$-\infty$增大到零时，其增益由 0 增至最大；然后当输入由 0 增加至$+\infty$时，其增益又由最大逐渐降低至 0，并总为正值。利用该函数可以使同一神经网络既能处理小信号，也能处理大信号。因为该函数的中间高增益区解决了处理小信号的问题，而在伸向两边的低增益区正好适用于处理大信号的输入。

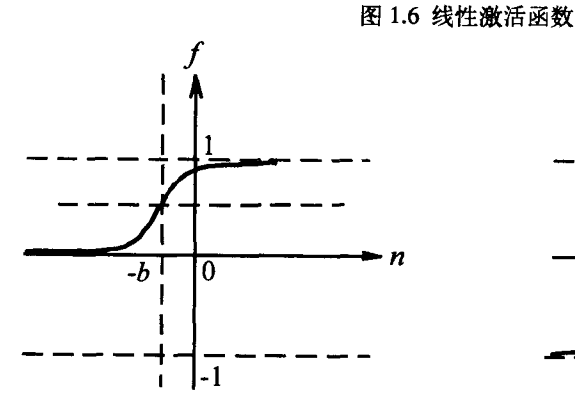
一般地，称一个神经网络是线性或非线性是由网络神经元中所具有的激活函数的线性或非线性来决定的。



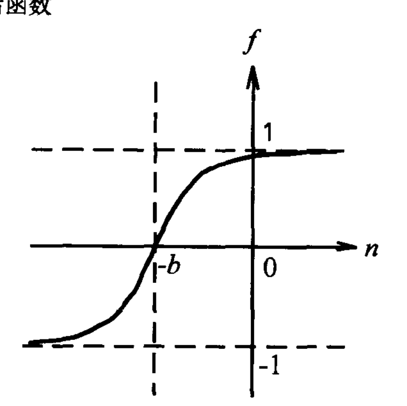
(a) 没有偏差的线性激活函数

(b) 带有偏差的线性激活函数

图 1.6 线性激活函数



(a) 带有偏差的对数 S 型激活函数



(b) 带有偏差的双曲正切型 S 型激活函数

图 1.7 S 型激活函数

1.5.3 单层神经元网络模型结构

将两个或更多的简单的神经元并联起来，使每个神经元具有相同的输入矢量 P ，即可组成一个神经元层，其中每一个神经元产生一个输出，图 1.8 给出一个具有 r 个输入分量，s 个神经元组成的单层神经元网络。

从结构图 1.8 中可以看出，输入矢量 P 的每个元素 p_j （j =1,2, ..., r），通过权矩阵 W 与每个输出神经元相连(即全联接)；每个神经元通过一个求和符号，在与输入矢量进行加权求和运算后，形成激活函数的输入矢量，并经过激活函数 $f(\cdot)$ 作用后得到输出矢量 A ，它可以表示为：

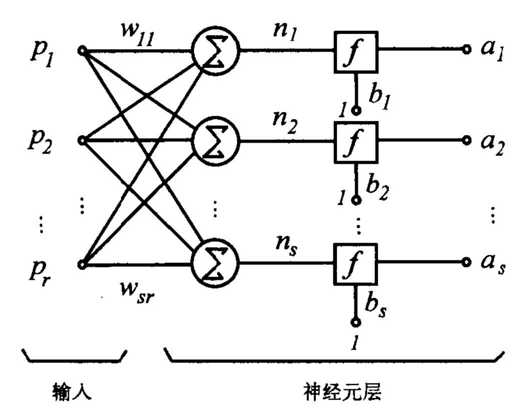


图 1.8 单层神经元网络模型结构

$$A_{s\times 1}=\mathrm{F}(W_{s\times r}*P_{r\times 1}+B_{s\times 1}) \tag{1.7}$$

其中，s 为神经元的个数，$F(\cdot)$ 表示激活函数。公式中的字母下标给出了矢量矩阵所具有的维数。一般情况下，输入分量数目 r 与层神经元数目 s 不相等，即 $s \neq r$ 。

网络权矩阵为：

$$W_{s\times r} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1r} \\ w_{21} & w_{22} & \cdots & w_{2r} \\ \cdots & \cdots & \cdots & \cdots \\ w_{s1} & w_{s2} & \cdots\cdot & w_{sr} \end{bmatrix}$$

注意，权矩阵 W 元素中的行表示神经元的位数，而列表示输入矢量的位数，因而，W_{12} 表示的是：来自第 2 个输入元素到第一个神经元之间的联接权值。

当有 q 组 r 个输入元素作为网络的输入时，输入矢量 P 则成为一个维数为 $r\times q$ 的矩阵：

$$P_{r\times q} = \begin{bmatrix} p_{11} & p_{12} & \cdots & w_{1q} \\ p_{21} & p_{22} & \cdots & p_{2q} \\ \cdots & \cdots & \cdots & \cdots \\ p_{s1} & p_{s2} & \cdots\cdot & w_{rq} \end{bmatrix}$$

此时的输出矢量为一个维数为 $s\times q$ 的矩阵 $\mathrm{A}_{s\times q}$：

$$A_{s\times q}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \cdots & \cdots & \cdots & \cdots \\ a_{s1} & a_{s2} & \cdots & a_{sq} \end{bmatrix}$$

1.5.4 多层神经网络

将两个以上的单层神经网络级联起来则组成多层神经网络。一个人工网络可以有许多层，每层都有一个权矩阵 W，一个偏差矢量 B 和一个输出矢量 A，为了对各层矢量矩阵加以区别，可以在各层矢量矩阵名称后加上层号来命名各层变量，例如，对第一层的权矩阵和输出矢量分别用 $W1$ 和 $A1$ 来表示，对第二层的这些变量表示为 $W2$ 和 $A2$ 等等依此类推。

一个三层的神经网络结构如图 1.9 所示。

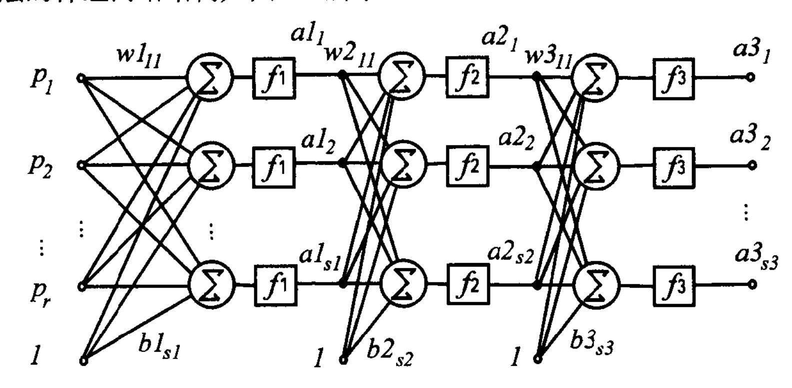


图 1.9　三层的神经网络结构图

图中所示网络结构具有 r 个输入矢量，第一层有 $s1$ 个神经元，第二层有 $s2$ 个神经元。一般情况下，不同层有不同的神经元数目，每个神经元都带有一个输入为常数 1 的偏差值。多层网络的每一层起着不同的作用，最后一层成为网络的输出，称为输出层，所有其他层称为隐含层。由此可见图 1.9 为具有两个隐含层的三层神经网络。有些设计者将与第一隐含层相连的输入矢量称为输入层，若以此来分，图 1.9 可称为四层神经网络，但网络中只有三组权矩阵。

为了便于简单明了地抓住神经网络各层的重点，我们将图 1.9 简化为图 1.10，并加上 q 组输入矢量。

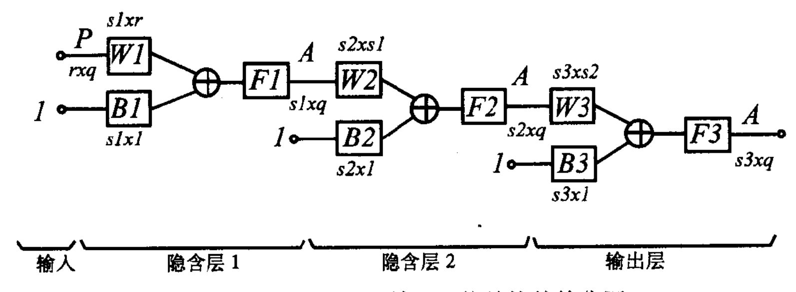


图 1.10　图 1.9 所示神经网络结构的简化图

在多层网络中，每一隐含层的输出都是下一层的输入，所以可以将层 2 看作一层具有 $s1 \times q$ 维输入矢量 $A1$，$s2 \times s1$ 维权矩阵 $W2$，以及 $s2 \times q$ 维输出矢量 $A2$ 的神经网络。既然已经分清了层 2 的所有矢量矩阵，则可以把它作为一个单层神经网络来处理，按照前节求法来计算输入/输出之间的函数关系。并利用此法可以写出任何一层网络输入/输出之间的对应关系式。

以此方式，图 1.10 所示神经网络的输出可以用下列数学式表示为：

$$
\begin{aligned}
A1 &= F1\ (W1*P+B1) \\
A2 &= F2\ (W2*A1+B2) \\
A3 &= F3\ (W3*A2+B3) \\
&= F3\left\{W3*F2\left[W2*F1\ (W1*P+B1)\ +B2\right]+B3\right\}
\end{aligned} \tag{1.8}
$$

对于这种标准全联接的多层神经网络，更一般的简便作图是仅画出输入节点和一组隐含层节点外加输出节点及其连线来示意表示，如图 1.11 所示。图中只标出输入、输出和权矢量，完全省去激活函数的符号。完整的网络结构则是通过具体的文字描述来实现的，如：网络具有一个隐含层，隐含层中具有 5 个神经元并采用 S 型激活函数，输出层采用线性函数，或者更简明的可采用“网络采用 2-5-1 结构”来描述，其中，2 表示输入节点数；5 表示隐含层节点数；1 为输出节点数。如果不对激活函数作进一步的说明，则意味着隐含层采用 S 型函数，而输出层采用线性函数。

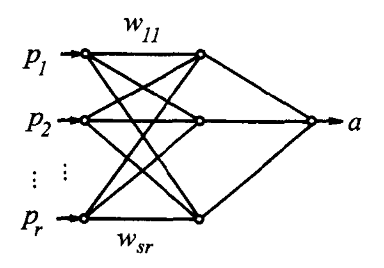


图 1.11　神经网络结构示意图

特别值得强调的是，在设计多层网络时，隐含层的激活网络应采用非线性的，否则多层网络的计算能力并不比单层网络更强。因为在采用线性激活函数的情况下，如果将偏差作为一组权值归为 W 中统一处理，两层网络的输出 $A2$ 则可以写为：

$$
\begin{aligned}
A2 &= F2(W2*A1) \\
&= W2*F1(W1*P) \\
&= W2*W1*P \\
&= W*P
\end{aligned} \tag{1.9}
$$

其中，$F1 = F2 = F$ 为线性激活函数。上式表明两层线性网络的输出计算等效于具有权矢量为 $W = W2*W1$ 的单层线性网络的输出。

1.5.5 反馈网络

人工神经网络按照网络拓扑结构可分为前向网络和反馈网络 (Recurrent Network) 两大类。前几节所见到的网络结构均为前向网络，其特点是：信号的流向是从输入通向输出。而反馈网络的主要不同点表现在它的输出信号通过与输入连接而返回到输入端，从而形成一个回路。一个具有 s 个神经元的典型的反馈网络如图 1.12 所示。

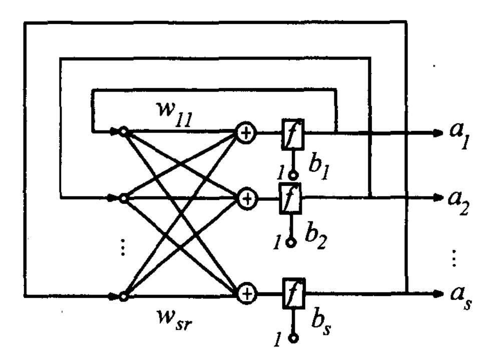


图 1.12 反馈网络结构图

前向网络的输出只是与当前输入及其联接权值有关，而在反馈网络中，由于将输出循环返回到输入，所以每一时刻的网络输出不仅取决于当前的输入，而且还取决于上一时刻的输出。其输出的初始状态由输入矢量设定后，随着网络的不断运行，从输出反馈到输入的信号不断改变，也使得输出不断变化，从而使网络表现出暂态特性，这使得反馈网络表现出前向网络所不具有的振荡或收敛特性。

1.6 用 MATLAB 计算人工神经网络输出

前面已经说过，表征一个神经网络特性的关键是网络的激活函数，对于多层网络可以将前一层的输出作为后一层的输入，而一层一层地计算直至网络的输出。在 MATLAB 环境下的神经网络工具箱中对于单层网络输出的计算，对于给出输入矩阵 P 、权矩阵 W 和偏差矩阵 B 的单层网络，只要简单地选用相应的激活函数，即可求出网络输出矩阵 A ，所有运算直接以矩阵形式进行。根据前几节所介绍的激活函数的种类，常用的单层神经元激活函数有:

1）输出为{0,1}的硬函数: **hardlim.m**

网络输出的计算公式为:

$$A = \text{hardlim}(\text{W*P}, \text{B});$$

2）输出为{-1,1}的二值函数: **hardlims.m**

网络输出的计算公式为:

$$A = \text{hardlims}(\text{W*P}, \text{B});$$

3）线性函数: **purelin.m**

网络输出的计算公式为:

$$A = \text{purelin}(\text{W*P}, \text{B});$$

4）对数 S 型函数: **logsig.m**

网络输出的计算公式为:

$$A = \text{logsig}(\text{W*P}, \text{B});$$

5）双曲正切 S 型函数: **tansig.m**

网络输出的计算公式为:

$$A = \text{tansig}(W*P, B);$$

下面我们从简单地计算网络的输出为例来看如何运用 MATLAB 工具箱解决问题的。

【例 1.1 】一个具有双曲正切 S 型激活函数的单层网络，输入矢量有 4 组，每组 3 个分量；输出矢量有 5 个神经元。假定输入矢量和权矢量均取值为 1，试用 MATLAB 计算网络的输出。

解:

```
%   outa.m               文件名
Q = 4;                   % 行数
R = 3;                   % 列数
S = 5;                   % 神经元数
W = ones ( S, R );       % 将数 1 赋予 SxR 维权矩阵 W
B = ones ( S, 1 );       % 将数 1 赋予 Sx1 维偏差矩阵 B
P = ones (R, Q );        % 将数 1 赋予 Rx1 维输入矩阵 P
flops ( 0 )              % 置 0
A = tansig ( W*P, B )    % 计算网络输出
flops                    % 计算并显示浮点操作次数
end                      % 程序结束
```

网络输出文件设计好后，将它存人名为 outa.m 的文件中，然后在 MATLAB 的工作界面的提示符 》后下写 outa 并打回车进行执行，则立刻可以看到 A 的输出结果以及运算所花费的浮点操作数:

```
》 outa
A =
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993

ans =
      280
```

这是由文件中的函数 **flops.m** 计算出的，它可以给出程序编写的运算过程所花费的浮点操作次数。用此函数可以在网络训练以及设计中，对所采用算法的优劣程度给出一个判断的依据。

在 MATLAB 环境下，用百分比号“ % ”表示注释；而每一语句后的分号“ ; ”表示在运行此命令后不显示其计算结果；若采用逗号“，”或无符号，则在运行后给出结果。诸如此类的编程规则在 MATLAB 的使用过程中能够很方便地随时通过 help 命令进行查询，而且每当编程有错时，程序运行后在工作界面里会指出来。

以上的 MATLAB 的.m 文件可以在工作界面的提示符下一句一句的写出并执行。不过笔者认为由于一般解题的程序不是靠一两句程序即可完成，从节省工作量以及便于操作考

虑，建议从对任何小的习题开始都养成写一个.m 文件的习惯，即将所做工作写进以扩展名为.m 的文件，这样可以随时修改、添加和删减其内容，比如上面【例 1.1】的第一行就用注释写出了本题的文件名为 outa.m。这个文件的生成只要在工作界面的右上角 File 中打开新的文件，即可开始编写自己的新程序。完成程序的编写后，将其另名保存为 outa.m 即可。运行时在工作界面写下 outa（扩展名可以省略不些）打回车。

【例 1.2】用 MATLAB 写出计算由图 1.9 所表示的三层神经网络的每层输出表达式。

已知：P = [0.1 0.5;
　　　　　0.3 -0.2];
　　　S1 = 2;　S2 = 3;　S3 = 5;

F1 为{-1, 1}二值型函数；F2 为对数 S 型函数；F3 为线性函数。

解：

```
% multout.m
P = [ 0.1   0.5; 0.3   -0.2 ];        % 已知输入矢量数据
S1 = 2;   S2 =3;   S3 = 5;            % 已知各层节点数
[ R, Q ] = size (P);                  % 求出输入矢量的行和列
[ W1, B1 ] = rands (S1, R);           % 给第一隐含层权值赋(-1, 1)之间的随机值
[ W2, B2 ] = rands (S2, S1);          % 给第二隐含层权值赋(-1, 1)之间的随机值
[ W3, B3 ] = rands (S3, S2);          % 给输出层权值赋(-1, 1)之间的随机值
A1 = hardlims ( W1*P, B1 )            % 计算第一层输出表达式
A2 = logsig ( W2*A1, B2 )             % 计算第二层输出表达式
A3 = purelin ( W3*A2, B3 )            % 计算输出层输出表达式
end
```

当把上述程序存入名为 multout.m 中并在工作界面运行后，可立刻得到在随机初始权值下的输出值 A1、A2 和 A3，这是因为程序中这三个表达式后没有加分号“;”的缘故。本例题中的实质就是三个输出表达式，其他内容均为赋值与参数初始化。由此可见应用 MATLAB 工具箱的简便。当计算复杂时，更能显示出它的优越性来。

参数初始化是神经网络设计中一项很重要的步骤，并且是由设计者来完成的。从【例 1.2】中可以看出，所有层中的权矩阵维数的大小均在随机取值函数 **rands.m** 运行中由所给定的网络各层的神经元 S1，S2 和 S3 确定的。三层网络中的每层网络权矩阵的初始化均是采用同一随机函数进行的，所不同的仅在于其中表示行和列的维数：第一层网络是 $S1 \times R$；第二层网络输出是：$s2 \times s1$；第三层网络输出是：$s3 \times s2$。由此可见，只有设计者对网络结构清楚明了后，才能够正确地写出其表达式，才能够运用好 MATLAB 工具箱快速准确地解决问题。

从下一章开始我们将逐一详细介绍各典型人工神经网络的模型结构、学习规则和训练过程，并结合实例，利用 MATLAB 环境下的神经网络工具箱，深入分析与探讨各类网络的特性、功能以及所存在的局限性，使读者能够更加全面地掌握人工神经网络的理论、设计与应用。

1.7 本章小结

1 ）人工神经网络是对生物神经网络进行仿真研究的结果，或者说，人工神经网络技术是根据所掌握的生物神经网络机理的基本知识，按照控制工程的思路和数学描述的方法，建立相应的数学模型，并采用适当的算法，有针对性地确定数学模型的参数（如联接权值、阈值等），以便获得某个特定问题的解；

2 ）经过几十年的兴衰，人们已经发展了上百种人工神经网络。但大部分网络都是几种典型网络的变形和组合。一般地说，人工神经网络的拓扑结构可分两种：前向网络和反馈网络。在本书中主要介绍和运用的典型的前向网络为：单层感知器、自适应线性网络和 BP 网络；反馈网络主要介绍和运用离散型与连续型霍普菲尔德网络，另外还将介绍的典型网络有：竞争网络和自适应共振理论（ ART ）网络；

3 ）学习人工神经网络的重点在于了解和掌握各典型人工神经网络的模型结构、学习规则和训练过程，以及灵活应用。神经网络的训练学习方式主要分为以下两种：监督式（也称有导师）的和无监督（也称为无导师）的。前者待分类的模式类别属性已知。对于每次模式样本的输入，网络输出端都有一个对应的指导（监督）信号与其属性相匹配。基于网络输出端监督信号与实际输出的某种目标函数准则，通过不断调整网络的联接权值，使得网络输出端的输出与监督信号的误差逐渐减小到预定的期望要求。而后者待分类的模式类别属性未知，网络结构和联接权值根据某种聚类法则，自动对周围环境的模式样本进行学习和调整，直至网络的结构及其联接分布能合理地反映训练样本的统计分布。所有这些具体内容都将在后面的章节里陆续进行详细阐述。

第二章　感知器

感知器是由美国计算机科学家罗森布拉特于1957年提出的。感知器可谓是最早的人工神经网络。单层感知器是一个具有一层神经元、采用阀值激活函数的前向网络。通过对网络权值的训练，可以使感知器对一组输入矢量的响应达到元素为0或1的目标输出。从而实现对输入矢量分类的目的。图2.1给出了单层感知器神经元模型图。

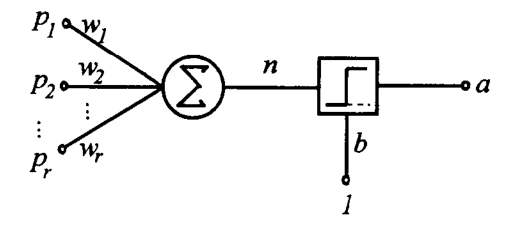


图2.1　感知器神经元模型

其中，每一个输入分量 $p_j(j=1,2,...,r)$ 通过一个权值分量 w_j 进行加权求和，并作为阀值函数的输入。偏差 b 的加入使得网络多了一个可调参数，为使网络输出达到期望的目标矢量提供了方便。感知器特别适合解决简单的模式分类问题。

感知器实际上是在MP模型的基础上加上学习功能，使其权值可以调节的产物。罗森布拉特研究了单层的以及具有一个隐含层的感知器。但在当时他只能证明单层感知器可以将线性可分输入矢量进行正确划分，所以本书中所说的感知器是指单层的感知器。多层网络因为要用到后面将要介绍的反向传播法进行权值修正，所以把它们均归类为反向传播网络之中。

2.1　感知器的网络结构

感知器的网络是由单层的 s 个感知神经元，通过一组权值 $\{w_{ij}\}$ ($i=1,2,...,s$; $j=1,2,...,r$)与 r 个输入相连组成。对于具有输入矢量 $P_{r\times q}$ 和目标矢量 $T_{s\times q}$ 的感知器网络的简化结构如图2.2所示。

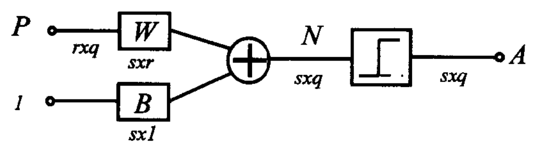


图2.2　感知器简化结构图

根据网络结构，可以写出第 i 个输出神经元（ $i=1,2,...,s$ ）的加权输入和 n_i 及其输出 a_i 为:

$$n_i = \sum_{j=1}^{r} w_{ij} p_j \tag{2.1}$$

$$a_i = f(n_i + b_i) \tag{2.2}$$

感知器的输出值是通过测试加权输入和值落在阈值函数的左右来进行分类的，即有：

$$a_i = \begin{cases} 1 & n_i + b_i \geqslant 0 \\ 0 & n_i + b_i < 0 \end{cases} \tag{2.3}$$

阈值激活函数如图 2.3 所示。

由图 2.3 可知：当输入 n_i+b_i 大于等于 0，即有 $n_i \geqslant -b_i$ 时，感知器的输出为 1，否则输出 a_i 为 0。利用偏差 b_i 的使用，使其函数可以左右移动，从而增加了一个自由调整变量和实现网络特性的可能性。

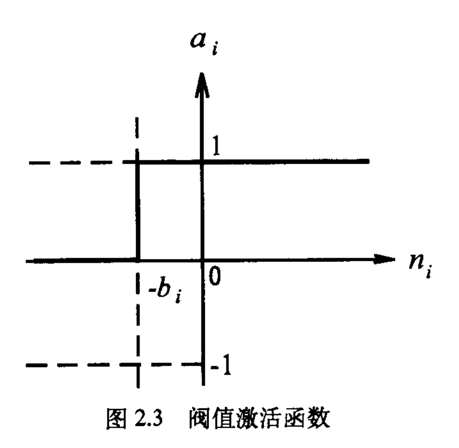


图 2.3　阈值激活函数

2.2　感知器的图形解释

由感知器的网络结构，我们可以看出感知器的基本功能是将输入矢量转化成 0 或 1 的输出。这一功能可以通过在输入矢量空间里的作图来加以解释。为了简单起见，以下取 $s=1$，即输出为一个节点的网络的情况来进行作图解释。

由感知器的输入/输出的关系式(2.3)可知，感知器的输出只有 1 或 0 两个状态，其他值由 $W*P+b$ 的值大于、等于或小于零来确定。当网络的权值 W 和 b 确定后，在由各输入矢量 p_j ($j=1, 2, ..., r$)为坐标轴所组成的输入矢量空间里，可以画出 $W*P+b=0$ 的轨迹，对于任意给定的一组输入矢量 P，当通过感知器网络的权值 W 和 b 的作用，或落在输入空间 $W*P+b=0$ 的轨迹上，或落在 $W*P+b=0$ 轨迹的上部或下部，而整个输入矢量空间是以 $W*P+b=0$ 为分割界，即不落在 $W*P+b=0$ 轨迹上的输入矢量，不是属于 $W*P+b>0$，就是使 $W*P+b<0$。因而感知器权值参数的设计目的，就是根据学习法则设计一条 $W*P+b=0$ 的轨迹，使其对输入矢量能够达到期望位置的划分。

以输入矢量 $r=2$ 为例，对于选定的权值 w_1、w_2 和 b，可以在以 p_1 和 p_2 分别作为横、纵坐标的输入平面内画出 $W*P+b=w_1p_1+w_2p_2+b=0$ 的轨迹，它是一条直线，此直线上的及其线以上部分的所有 p_1、p_2 值均使 $w_1p_1+w_2p_2+b \geqslant 0$，这些点若通过由 w_1、w_2 和 b 构成的感知器则使其输出为 1；该直线以下部分的点则使感知器的输出为 0。所以当采用感知器对不同的输入矢量进行期望输出为 0 或 1 的分类时，其问题可转化为：对于已知输入矢量在输入空间形成的不同点的位置，设计感知器的权值 W 和 b，将由 $W*P+b=0$ 的直线放置在适当的位置上使输入矢量按期望输出值进行上下分类。

阈值函数通过将输入矢量的 r 维空间分成若干区域而使感知器具有将输入矢量分类的能力。输出矢量的 0 或 1，取决于对输入的分类。

图 2.4 给出了感知器在输入平面中的图形，从中可以清楚看出：由直线 $W*P+b=0$ 将由输入矢量 p_1 和 p_2 组成的平面分为两个区域，此线与权重矢量 W 正交可根据偏差 b 进行左右平移。直线上部的输入矢量使阀值函数的输入大于 0，所以使感知器神经元的输出为 1。直线下部的输入矢量使感知器神经元的输出为 0。分割线可以按照所选的权值和偏差上下左右移动到期望划分输入平面的地方。

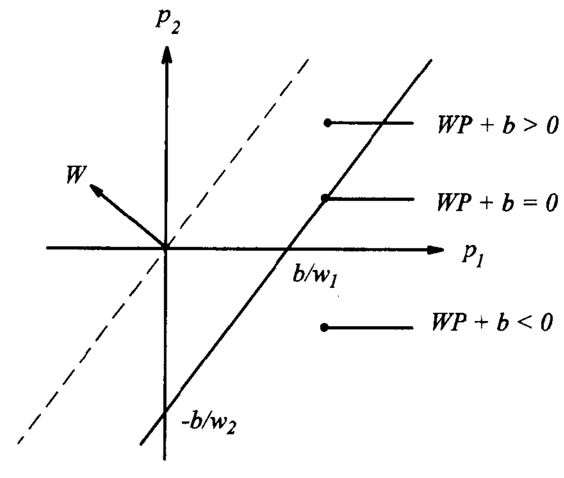


图 2.4　输入矢量平面图

感知器神经元不带偏差时，得到的是通过原点的分类线。有些可以用带偏差解决的问题，不带偏差的网络则解决不了。

熟悉图形解释有助于我们理解和掌握感知器的工作原理。当然在实际应用时，权值的求解全都是由计算机来完成的。

2.3　感知器的学习规则

学习规则是用来计算新的权值矩阵 W 及新的偏差 B 的算法。感知器利用其学习规则来调整网络的权值，以便使该网络对输入矢量的响应达到数值为 0 或 1 的目标输出。

对于输入矢量 P，输出矢量 A，目标矢量为 T 的感知器网络，感知器的学习规则是根据以下输出矢量可能出现的几种情况来进行参数调整的。

1）如果第 i 个神经元的输出是正确的，即有：$a_i=t_i$，那么与第 i 个神经元联接的权值 w_{ij} 和偏差值 b_i 保持不变；

2）如果第 i 个神经元的输出是 0，但期望输出为 1，即有 $a_i=0$，而 $t_i=1$，此时权值修正算法为：新的权值 w_{ij} 为旧的权值 w_{ij} 加上输入矢量 p_j；类似的，新的偏差 b_i 为旧偏差 b_i 加上它的输入 1；

3）如果第 i 个神经元的输出为 1，但期望输出为 0，即有 $a_i=1$，而 $t_i=0$，此时权值修正算法为：新的权值 w_{ij} 等于旧的权值 w_{ij} 减去输入矢量 p_j；类似的，新的偏差 b_i 为旧偏差 b_i 减去 1。

由上面分析可以看出感知器学习规则的实质为：权值的变化量等于正负输入矢量。具体算法总结如下。

对于所有的 i 和 j, $i = 1, 2, \ldots, s$； $j = 1, 2, \ldots, r$，感知器修正权值公式为：

$$\begin{aligned} \Delta w_{ij} &= (t_i - y_i) \times p_j \\ \Delta b_i &= (t_i - y_i) \times 1 \end{aligned} \tag{2.4}$$

用矢量矩阵来表示为：

$$\begin{aligned} W &= W + EP^T \\ B &= B + E \end{aligned} \tag{2.5}$$

此处，E 为误差矢量，有 $E = T - A$ 。

感知器的学习规则属于梯度下降法，该法则已被证明：如果解存在，则算法在有限次的循环迭代后可以收敛到正确的目标矢量。

上述用来修正感知器权值的学习算法在 MATLAB 神经网络工具箱中已编成了子程序，成为一个名为 **learnp.m** 的函数。只要直接调用此函数，即可立即获得权值的修正量。此函数所需要的输入变量为：输入、输出矢量和目标矢量：P、A 和 T。调用命令为：

```
[ dW, dB ] = learnp ( P, A, T );
```

2.4 网络的训练

要使前向神经网络模型实现某种功能，必须对它进行训练，让它逐步学会要做的事情，并把所学到的知识记忆在网络的权值中。人工神经网络权值的确定不是通过计算，而是通过网络的自身训练来完成的。这也是人工神经网络在解决问题的方式上与其他方法的最大不同点。借助于计算机的帮助，几百次甚至上千次的网络权值的训练与调整过程能够在很短的时间内完成。

感知器的训练过程如下：

在输入矢量 P 的作用下，计算网络的实际输出 A，并与相应的目标矢量 T 进行比较，检查 A 是否等于 T，然后用比较后的误差 E，根据学习规则进行权值和偏差的调整；重新计算网络在新权值作用下的输入，重复权值调整过程，直到网络的输出 A 等于目标矢量 T 或训练次数达到事先设置的最大值时训练结束。

若网络训练成功，那么训练后的网络在网络权值的作用下，对于被训练的每一组输入矢量都能够产生一组对应的期望输出；若在设置的最大训练次数内，网络未能够完成在给定的输入矢量 P 的作用下，使 $A = T$ 的目标，则可以通过改用新的初始权值与偏差，并采用更长训练次数进行训练，或分析一下所要解决的问题是否属于那种由于感知器本身的限制而无法解决的一类。

感知器设计训练的步骤可总结如下：

1）对于所要解决的问题，确定输入矢量 P，目标矢量 T，并由此确定各矢量的维数以及确定网络结构大小的神经元数目：r，s 和 q；

2）参数初始化：

a) 赋给权矢量 W 在(-1, 1)的随机非零初始值;

b) 给出最大训练循环次数 max_epoch ;

3 ）网络表达式：根据输入矢量 P 以及最新权矢量 W，计算网络输出矢量 A ;

4 ）检查：检查输出矢量 A 与目标矢量 T 是否相同，如果是，或已达最大循环次数，训练结束，否则转入 5 ）;

5 ）学习：根据(2.5)式感知器的学习规则调整权矢量，并返回 3)。

下面给出例题来进一步了解感知器解决问题的方式，掌握设计训练感知器的过程。

【例 2.1】考虑一个简单的分类问题。

设计一个感知器，将二维的四组输入矢量分成两类。

输入矢量为：P = [-0.5　-0.5　0.3　0;

　　　　　　　　-0.5　0.5　-0.5　1];

目标矢量为：T = [1.0　1.0　0　0],

解:

通过前面对感知器图解的分析可知，感知器对输入矢量的分类实质是在输入矢量空间用 $W*P + B = 0$ 的分割界对输入矢量进行切割而达到分类的目的。根据这个原理，对此例中二维四组输入矢量的分类问题，可以用下述不等式组来等价表示出:

$$\begin{cases} -0.5w_1 - 0.5w_2 + w_3 \geqslant 0 & (\text{使 } t_1 = 1 \text{ 成立}) \\ -0.5w_1 + 0.5w_2 + w_3 \geqslant 0 & (\text{使 } t_2 = 1 \text{ 成立}) \\ 0.3w_1 - 0.5w_2 + w_3 < 0 & (\text{使 } t_3 = 0 \text{ 成立}) \\ w_2 + w_3 < 0 & (\text{使 } t_4 = 0 \text{ 成立}) \end{cases}$$

实际上可以用代数求解法来求出上面不等式中的参数 w_1 、 w_2 和 w_3。经过迭代和约简，可得到解的范围为:

$$\begin{cases} w_1 < 0 \\ 0.8w_1 < w_2 < -w_1 \\ w_1/3 < w_3 < -w_1 \\ w_3 < -w_2 \end{cases}$$

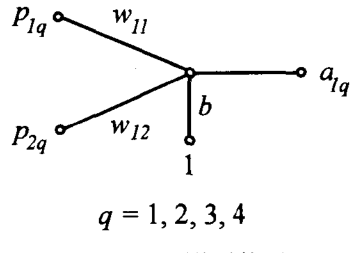
一组可能解为:

$$\begin{cases} w_1 = -1 \\ w_2 = 0 \\ w_3 = -0.1 \end{cases}$$

而当采用感知器神经网络来对此题进行求解时，意味着采用具有阀值激活函数的神经网络，按照问题的要求设计网络的模型结构，通过训练网络权值 $W = [w_{11}\ w_{12}]$和 b，并根据学习算法和训练过程进行程序编程，然后运行程序，让网络自行训练其权矢量，直至达到不等式组的要求。

鉴于输入和输出目标矢量已由问题本身确定，所以所需实现其分类功能的感知器网络结构的输入节点 r，以及输出节点数 s 已被问题所确定而不能任意设置。

根据题意，网络结构图如图 2.5 所示。



$q = 1, 2, 3, 4$

图 2.5　网络结构图

由此可见，对于单层网络，网络的输入神经元数 r 和输出神经元数 s 分别由输入矢量 P 和目标矢量 T 唯一确定。网络的权矩阵的维数为：$W_{s\times r}$，$B_{s\times 1}$权值总数为 $s \times r$ 个，偏差个数为 s 个。

在确定了网络结构并设置了最大循环次数和赋予权值初始值后，设计者可方便地利用 MATLAB，根据题意以及感知器的学习、训练过程来编写自己的程序。下面是对【例 2.1】所编写的网络权值训练用的 MATLAB 程序：

```
% percep1.m
%
P = [ -0.5 -0.5 0.3 0; -0.5 0.5 -0.5 1 ];          T = [ 1 1 0 0 ];
% 初始化
[ R, Q ] = size (P);        [S, Q ] =size (T);
W = rands ( S, R );         B = rands ( S, 1 );
max_epoch = 20;
% 表达式
A = hardlim ( W*P, B );                 % 求网络输出
for epoch = 1 : max_epoch               % 开始循环训练、修正权值过程
% 检查
  if all ( A == T )                     % 当 A = T 时结束
     epoch = epoch - 1;
     break
  end
% 学习
[ dW, dB ] = learnp ( P, A, T );        % 感知器学习公式
W = W + dW;
B = B + dB;
A = hardlim ( W*P, B );                 % 计算权值修正后的网络输出
end                                     % 程序结束
```

以上就是根据前面所阐述的感知器训练的三个步骤：表达式、检查和学习而编写的 MATLAB 网络设计的程序。

对于本例也可以在二维平面坐标中给出求解过程的图形表示。图 2.6 给出了横轴为 p_1，纵轴为 p_2 的输入矢量平面，以及输入矢量 P 所处的位置。根据目标矢量将期望为 1 输出的输入分量用 "+" 表示出，而目标为 0 输出的输入分量用 "o" 表示出。

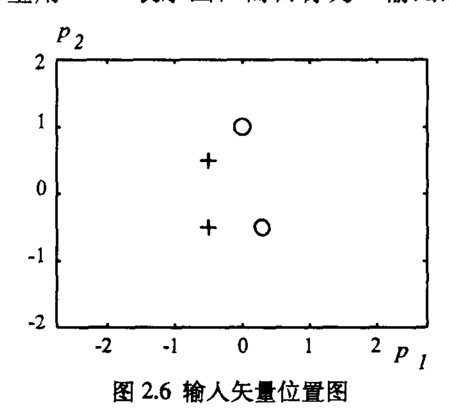


图 2.6 输入矢量位置图

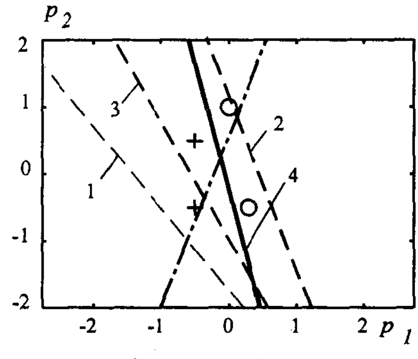


图 2.7 感知器训练过程记录

图 2. 7 给出 4 次训练过程中由权值组成的 $W*P + b = 0$ 直线和最终的训练达到目标后的直线，其中，点划线为初始权值形成的直线；虚线为前三次权值修正后划分的结果；显然，"o"和"+"部分没有被合适的分割开。实线是第 4 次权值修正后达到目标的结果。

本例题中网络初始随机权值为:

W0 = [-0.8161　0.3078];　 B0 = [-0.1680];

4 次训练次后的权值为:

W = [-2.6161 -0.6922];　B = -0.1680 ;

只要用下列命令即可得到对网络结果的验证:

```
》 A = hardlim ( W*P, B )
A =
  1 1 0 0
```

图 2.8 给出网络训练过程中的权值和偏差变化记录，其中，横轴变量为训练次数 epoch 。

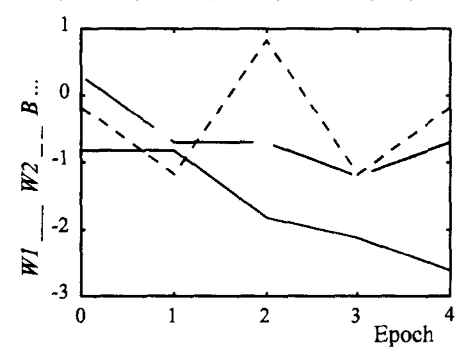


图 2.8 权值和偏差变化记录

感知器对于不同的初始条件可能得到不同的结果。对【例 2.1 】用不同的初始条件重新运行程序进行训练，同样可以解决问题，但得到不同于前一次的权值，例如对于下列权矢量和偏差值:

W = [-2.1642 -0.0744]

B = -0.6433

也是网络的一个解,但在输入平面中对输入矢量进行划分的直线 $W*P+B=0$ 的位置与前面的是不同的。

【例 2. 2 】多个神经元分类，又称模式联想。

若将【例 2. 1 】的输入矢量和目标矢量增加为 10 组的二元矩阵，即输入矢量为:

$P=[0.1\ \ 0.7\ \ 0.8\ \ 0.8\ \ 1.0\ \ 0.3\ \ 0.0\ \ -0.3\ \ -0.5\ \ -1.5;$
$\quad 1.2\ \ 1.8\ \ 1.6\ \ 0.6\ \ 0.8\ \ 0.5\ \ 0.2\ \ 0.8\ \ -1.5\ \ -1.3\ ];$

所对应的 10 组二元目标矢量为:

$T=[1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0;$
$\quad 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1];$

由于增加了矢量数组，必然增加了解决问题的复杂程度。这个问题要是用一般的方法来解决是相当困难和费时的，它需要解一个具有 6 个变量的 20 个约束不等式。而通过感知器来解此问题就显示出它的优越性。完全与【例 2.1 】程序相同，只要输入新的 P 和 T，重新运行程序，即可得出训练的结果。

根据输入矢量和目标矢量可得: $r=2, s=2$, 由此可以画出网络结构如图 2.9 所示。

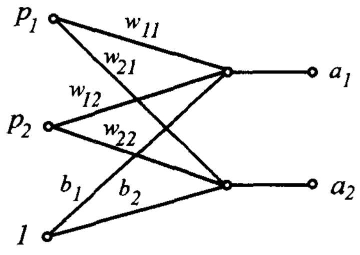


图 2.9　所设计网络结构图

本题用图解法解释为: 求感知器网络权值 W 和 B，使由 $W*P+B=0$ 构成的两条直线将输入矢量所在平面分割成四个区域。图 2.10 给出了输入矢量与不同的目标矢量所对应的不同关系，不同的目标矢量分别用四种符号表示: 00: o; 01: *; 10: +和 11: ×。

虽然【例 2.2 】的求解过程复杂了许多，但用 MATLAB 来对其进行训练和设计与【例 2.1 】没有什么区别。若我们再利用工具箱的特长，则显得更加简单。在神经网络工具箱中有一个集感知器训练过程中的表达式、检查和权值学习为一体的函数: **trainp.m** 。通过此函数，可直接给出最终的训练结果 W 和 B 值，以及所花费的循环次数 epochs 。这就允许我们腾出精力来增加一些实时监视程序或作图程序，从而使得训练过程更加生动。【例 2.2 】的训练程序可编写如下:

```
% percep2.m
%
% 初始化、赋值
P = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5; 1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
T = [ 1 1 1 0 0 1 1 1 0 0 ; 0 0 0 0 0 1 1 1 1 1 ];
[ R, Q ] = size (P);  [ S, Q ] = size (T);  [ W0, B0 ] = rands ( S, R );
disp_freq = 1;                % 每训练一次，显示一次
max_epoch = 20;               % 设置最大循环次数
TP = [ disp_freq   max_epoch ];   % 给 TP 赋值
% 训练网络、修正权值
[ W, B, epochs ] = trainp ( W0, B0, P, T, TP )
```

```
% 绘制训练后的分类结果
V = [ -2 2 -2 2 ];                % 取一数组限制坐标数值大小
plotv ( P, T, V );                % 在输入矢量空间绘画输入矢量和目标矢量的位置
axis ('equal'),                   % 令横坐标和纵坐标等距离长度
title ('Input Vector Graph'),     % 写图标题
xlabel ('p1'),                    % 写横轴标题
ylabel ('p2'),                    % 写纵轴标题
hold on                           % 当前图形保护模式开
plotpc ( W0, B0,'-' );            % 绘制由 W0 和 B0 在输入平面中形成的初始分类线
plotpc ( W, B );                  % 绘制由 W 和 B 在输入平面中形成的最终分类线
hold off                          % 当前图形保护模式关闭
end
```

当此程序运行后，每修正一次权值后，在工作界面中显示一次已循环的次数，并在训练完成后显示出最终权值 W 和 B 以及所花费的循环总数。另外还绘制出输入矢量在输入平面中的位置以及由训练后的 W 和 B 值组成的 $W*P+B=0$ 直线在输入平面中的分类结果。

经过 10 次训练调整神经元的权矢量，网络将输入矢量分成期望的四类如图 2.11 所示。其中虚线为初始值，实线为最后的结果. 网络最终权矢量为:

```
W = [ -4.7926 5.9048;
      -3.2567 -2.6339];
B = [-0.9311; 2.9970]
```

所对应的网络初始权值为:

```
W0 = [-0.6926 0.6048; 0.1433 -0.9339];      B0 = [0.0689; 0.0030];
```

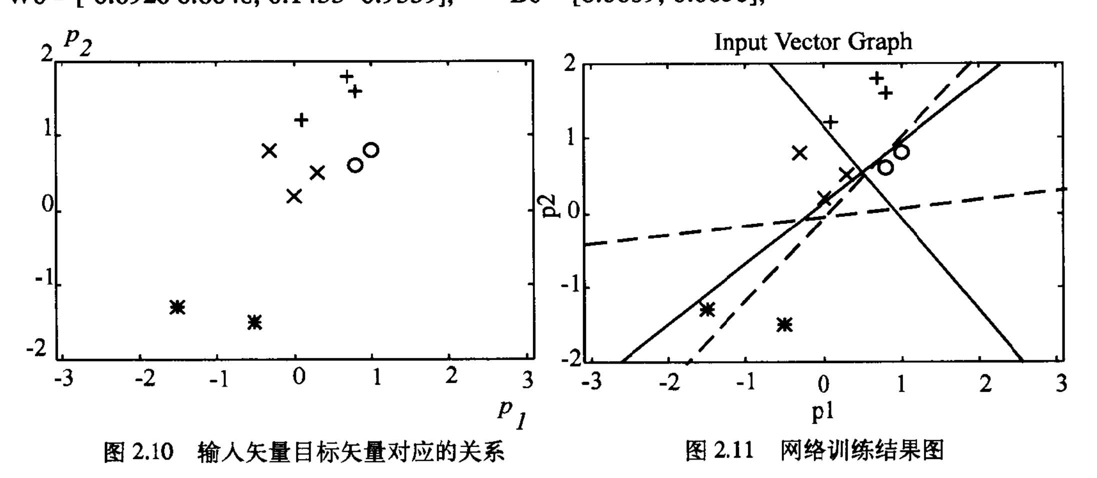


图 2.10　输入矢量目标矢量对应的关系　　　　图 2.11　网络训练结果图

对于由不同的输入神经元 r 和输出神经元 s 组成的感知器，当采用输入矢量空间的作

图法来解释网络功能时，其一般情况可以总结如下:

1) 当输入为单个元素，输出也为单个神经元，即 $r=1$， $s=1$ 时，感知器是以点为分割界;

2) 当输入为二元素，即 $r=2$ 时，感知器是以线为分割界;

其中： 当 $s=1$，分割线为一条线;

$s=2$，分割线为两条线。

依此类推。在这里输出神经元数 s 决定分割线数，可分成的种类数为 2^s;

3) 当输入为三元素，即 $r=3$ 时，感知器是以面为分割界，而且输出神经元数 s 为分割面数。

2.5 感知器的局限性

由于感知器自身结构的限制，使其应用被限制在一定的范围内。所以在采用感知器解决具体问题时，必须时刻考虑到其特点。一般来说，感知器有以下局限性:

1) 由于感知器的激活函数采用的是阀值函数，输出矢量只能取 0 或 1，所以只能用它来解决简单的分类问题;

2) 感知器仅能够线性地将输入矢量进行分类。如果用一条直线或一个平面把一组输入矢量正确地划分为期望的类别，则称该输入/输出矢量是对线性可分的，否则为线性不可分。那么，利用感知器将永远也达不到期望输出的网络权矩阵。所以用软件设计感知器对权值进行训练时，需要设置一个最大循环次数。如果在达到该最大循环次数后，还没有达到期望的目标，训练则停止，以便不使不可分的矢量占用无限循环的训练时间。不过应当提醒的是，理论上已经证明，只要输入矢量是线性可分的，感知器在有限的时间内总能达到目标矢量;

3) 感知器还有另外一个问题，当输入矢量中有一个数比其他数都大或小得很多时，可能导致较慢的收敛速度。比如当输入/输出矢量分别为:

P = [-0.5 -0.5 +0.3 -0.1 -80;
      -0.5 +0.5 -0.5 +1.0 100];
T = [1 1 0 0 1];

由于输入第五组数远远大于其他输入数组，这必然导致训练的困难。

下面给出感知器面对线性不可分输入/输出模式时的情况。

【例 2.3】线性不可分输入矢量。

由于感知器对输入矢量空间只能线性地进行输出 0 或 1 的分类，它们只能适当地划分具有线性可分的输入矢量组。如果在输入矢量组和它所对应的 0， 1 目标之间不能用直线进行划分，那么感知器就不能正确地分类出输入矢量。

在【例 2.1】中，加入一个新的输入矢量，使之成为:

输入矢量为: P = [-0.5 -0.5 0.3 0 -0.8;
                  0.5 0.5 -0.5 1 0];

目标矢量为: $T = [1.0 \quad 1.0 \quad 0 \quad 0 \quad 0]$;

可以重复使用前面的程序。为了能够让程序自己判断出所做训练是否有效，可以在程序中增加判断显示程序。这只需要在初始化阶段加入函数 **flops.m**，然后在程序最后加上:

```
fprintf ('\n Final Network Values : \n' )
W
B
fprintf ('Trained for %.0f epochs. \n', epoch )
fprintf ('Training took %.0f flops. \n', flops )
fprintf ('Nwtwork classifies :' );
if all ( hardlim ( W*P, B ) == T )
  disp ('Correctly.' )
else
  disp ('Incorrectly.' )
end
```

图 2.12 给出了要分类的数据组的初始值和训练 40 后的结果。从中可以看出，网络没有能够解决问题。图中实线的结果为:

$$W = [-2.1912 \quad -1.2975], B = -1.4214$$

对应网络训练的初始权值为:

$$W0 = [0.9088\ 0.7025]; \qquad B0 = [-0.4214]$$

检验这组由训练权矢量所构成的感知器的输出分类功能，用命令:

```
» A = hardlim ( W*P, B )
A =
  1 0 0 0 1
```

而 T = [1 1 0 0 0]。很显然可以看到在第二和第五个输入矢量上为不正确的分类。

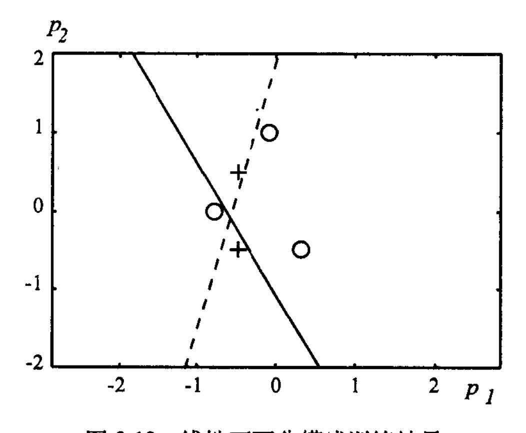


图 2.12　线性不可分模式训练结果

在感知器的应用中，这类线性不可分问题都可归结为“异或”问题。下面就来具体讨论一下这个问题。

2.6 “异或”问题

明斯基等人通过对单层感知器的研究得出结论：单层感知器存在着局限性。他的主要论据之一就是感知器不能实现简单的“异或”逻辑功能。

逻辑运算中的“异或”功能，就是当输入两个二进制数（0或1）均为0或1时，输出为0，只有当两个输入中有一个为1时，输出为1。若希望用感知器来实现此功能的操作，其输入应为两个二进制数的四种组合，目标输出T为与输入对应的上述四种输出的组合，即

$$P=[0\ 0\ 1\ 1;\quad 0\ 1\ 0\ 1];$$

$$T=[\ 0\ 1\ 1\ 0];$$

所要求解的是：设计一个单层感知器对输入矢量按期望的输出矢量进行分类。

我们已经知道，当输入为二元素时，其分割界为直线，可以用来进行划分的直线是数目与感知器输出神经元的数相等。“异或”问题则是要求用一条直线将平面上的四个点分成两类。而此四点在输入矢量平面上的位置如图2.13所示，其中，“×”表示希望将其点分为1类；“o”表示目标输出为0类。

很显然，对于这样的四点位置，想用一条直线把同类期望输入区分开是不可能的。实际上，如果我们把工作做得再详细点，从逻辑运算入手，即列出感知器对所有的四组二元素输入的可能输出的情况，一共可得16种情况，可以分别代表“与”、“或”、“非”、“与非”、“或非”、“异或”、“异或非”等逻辑功能。用一条直线，将平面分成两部分，分别对16种情况进行划分。结果是，16种功能中只有“异或”和“异或非”是线性不可分的，其他14种逻辑功能均为线性可分，它们都可以用单层感知器来实现。

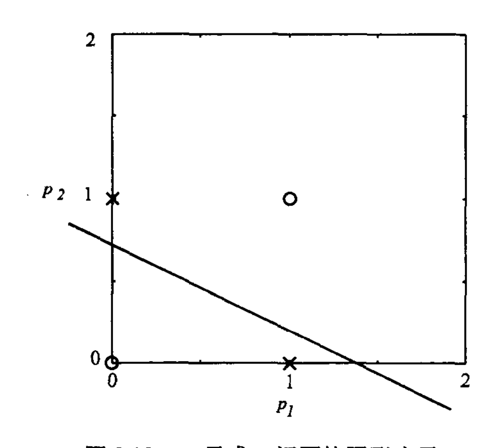


图2.13 “异或”问题的图形表示

输入矢量:

$$P=[0\,0\,1\,1;\;0\,1\,0\,1];$$

16 种目标矢量:

$$T1=[0\,0\,0\,0]$$
$$T2=[0\,0\,0\,1]$$
$$T3=[0\,0\,1\,0]$$
$$\cdots$$
$$T6=[0\,1\,0\,1]$$
$$T7=[0\,1\,1\,0] \quad \text{— 异或}$$
$$\cdots$$
$$T9=[1\,0\,0\,0]$$
$$T10=[1\,0\,0\,1] \quad \text{— 异或非}$$
$$\cdots$$
$$T16=[1\,1\,1\,1]$$

我们不禁要问：在有限的逻辑运算中，哪些是线性可分的呢？

我们再来考察一下最简单的单个元素输入感知器时可能有的所有逻辑功能。此时，输入有两种可能，输出有四种情况，即

输入矢量:

$$P=[0\quad 1];$$

4 种目标矢量:

$$T1=[0\,0];$$
$$T2=[0\,1];$$
$$T3=[1\,0];$$
$$T4=[1\,1];$$

因为只有一个输入，其分割界为以输入 P 为坐标线上的点，四种功能可表示为坐标轴上在 0 和 1 点上的不同功能的组合如图 2.14 所示。我们用“o”表示 0，“x”表示 1。则四种功能可表示为:

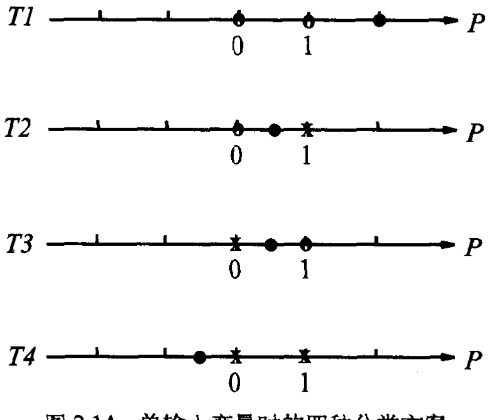


图 2.14　单输入变量时的四种分类方案

所要做的是，在 P 坐标上选一点，图中用黑点表示，将每一功能分成两类。这实际上是要分别解四组不同的分类问题。每一次分类中有一个 w 和一个 b，通过两个约束不等式来进行调节。所以这是个完全有解的线性代数问题，实际上，从图示中很明显地可以看出，只要调整 w 和 b 就很容易找到满足目标矢量的分界点。

同理，当输入具有 3 个元素时，可构成 8 种不同组合，此时的期望目标 T 可有 2 5 6 种功能。研究表明，其中只有 1 0 4 种是线性可分的。

从以上的例子可以得出结论，当网络具有 r 个二进制输入分量时，最大不重复的输入矢量有 2^r 组， 其输出矢量所能代表的逻辑功能总数为 2^{2^r}，表 2.1 给出了不同输入 r 时，线性可分的功能数。

表 2.1 不同输入 r 时线性可分的功能数

r	2^{2^r}	线性可分功能数
1	4	4
2	16	14
3	256	104
4	65536	1882
5	4.3×10^9	94572
6	1.8×10^{19}	5028134

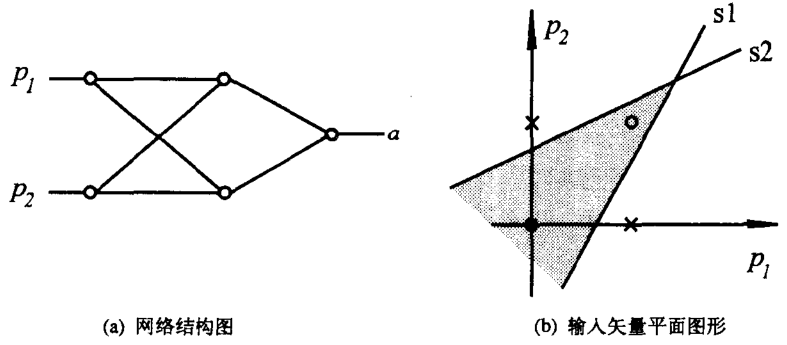
由表 2.1 可知，对于给定输入矢量所设计出的单层感知器，只对一部分输出功能是线性可分的。随着 r 的增加，线性不可分的功能数急剧增加。因此，当给定一个输入/输出矢量对时，首先必须判别该功能是否是线性可分的。遗憾的是，至今为止并没有办法来判定这种线性可分性。尤其当输入矢量增多时，更是难以确定。一般只有通过用一定的循环次数对网络进行训练而判定它是否能被线性可分。所以，单层神经元感知器只能用于简单的分类问题。

2.7 解决线性可分性限制的办法

感知器的线性可分性限制是个严重的问题。6 0 年代末，人们曾致力于该问题的研究，并找到了解决问题的办法，即变单层网络结构为多层网络结构。这实际上是把感知器的概念扩展化了。这样对于“异或”问题，我们可以用两层网络结构，并在隐含层中采用两个神经元，即用两条判决直线 s1 和 s2 来解决，如图 2.15 所示。

使 s1 线下部分为 1，线上部分为 0，而 s2 线的上部为 1，下部为 0，而在输出层中使图中阴影部分为 0，即可使“异或”功能得以实现，而且其实现的可能有许多种。研究表明，两层的阀值网络可以实现任意的二值逻辑函数，且输入值不仅限于二进制数，可以是连续数值。

对于多层感知器的权值训练与学习则需要用到误差的反向传播法，即后面将要学习的 BP 算法。本书中将具有隐含层的网络归为反向传播网络类而不再称为多层感知器。



(a) 网络结构图　　(b) 输入矢量平面图形

图 2.15 “异或”问题的一种解决方案

2.8 本章小结

1）感知器在分类问题上是有效的。它可以很好的划分线性可分的输入矢量，感知器网络的设计完全由需要解决的问题所限制。具有单层的阀值神经网络，其输入节点和神经元数是由输入矢量和输出矢量所确定的;

2）训练时间对突变矢量是敏感的。但突变输入矢量不妨碍网络达到目标。

感知器在解决实际问题时，必须在输入矢量是线性可分时才有效，这是很难得到的情形。虽然感知器有上述局限性，但它在神经网络研究中有着重要的意义和地位。它提出了自组织自学习的思想。对能够解决的问题有一个收敛的算法，并从数学上给出了严格的证明。对这种算法性质的研究仍是至今存在的多种算法中最清楚的算法之一。因此它不仅引起了众多学者对人工神经网络研究的兴趣，推动了人工神经网络研究的发展，而且后来的许多种网络模型都是在这种指导思想下建立起来并改进推广的。

习　题

〖2.1〗利用 MATLAB 对下列具有突变矢量进行编程以观察其对收敛速度的影响。

$$P = [\,-0.5\ -0.5\ +0.3\ -0.1\ -80;\quad -0.5\ +0.5\ -0.5\ +1.0\ 100];$$

$$T = [\,1\quad 1\quad 0\quad 0\quad 1];$$

〖2.2〗采用单层感知器的权值训练公式，通过固定隐含层的目标输出为线性可分矢量，设计一个两层感知器来解决“异或”问题，隐含层用两个神经元。

第三章　自适应线性元件

自适应线性元件(Adaptive Linear Element，简称 Adaline)也是早期神经网络模型之一，它是由威德罗和霍夫首先提出的。它与感知器的主要不同之处在于其神经元有一个线性激活函数，这允许输出可以是任意值，而不仅仅只是像感知器中那样只能取 0 或 1 。另外，它采用的是 W-H 学习法则，也称最小均方差(LMS)规则对权值进行训练，从而能够得到比感知器更快的收敛速度和更高的精度。

自适应线性元件的主要用途是线性逼近一个函数式而进行模式联想。另外，它还适用于信号处理滤波、预测、模型识别和控制。

3.1　自适应线性神经元模型和结构

一个线性的具有 r 个输人的自适应线性神经元模型如下图所示。这个神经元有一个线性激活函数，被称为 Adaline 如图 3.1(a)所示。和感知器一样，偏差可以用来作为网络的一个可调参数，提供额外可调的自由变量以获得期望的网络特性。线性神经元可以训练学习一个与之对应的输入/输出的函数关系，或线性逼近任意一个非线性函数，但它不能产生任何非线性的计算特性。

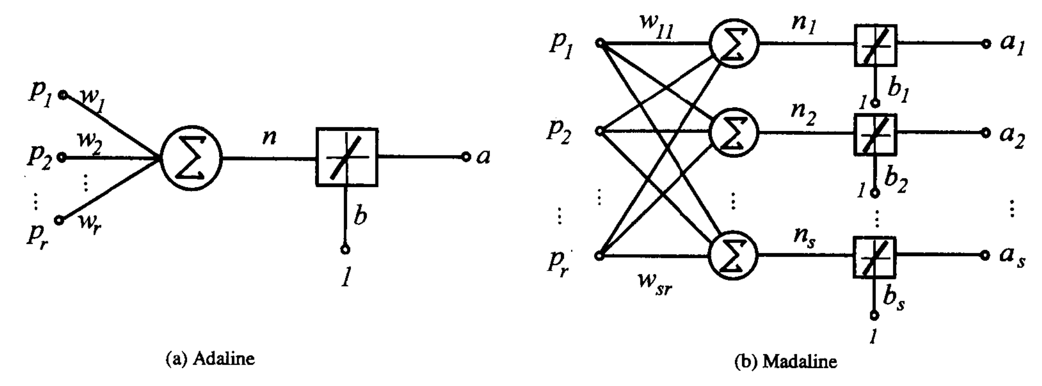


图 3.1　自适应线性神经网络的结构

当自适应线性网络由 s 个神经元相并联形成一层网络，此自适应线性神经网络又称为 Madaline 如图 3.1(b)所示。

W-H 规则仅能够训练单层网络，但这并不是什么严重问题。如第一章所述，单层线性网络与多层线性网络具有同样的能力，即对于每一个多层线性网络，都具有一个等效的单层线性网络与之对应。在反向传播法产生以后，威德罗又将其自适应线性网络扩展成多层，甚至将激活函数也扩展成非线性的了。

3.2 W-H 学习规则

W-H 学习规则是由威德罗和霍夫提出的用来修正权矢量的学习规则，所以用他们两人姓氏的第一个字母来命名。采用 W-H 学习规则可以用来训练一层网络的权值和偏差使之线性地逼近一个函数式而进行模式联想（ Pattern Association ）。

定义一个线性网络的输出误差函数为:

$$E(W,B)=\frac{1}{2}[T-A]^2=\frac{1}{2}[T-WP]^2 \tag{3.1}$$

由(3.1)式可以看出：线性网络具有抛物线型误差函数所形成的误差表面，所以只有一个误差最小值。通过 W-H 学习规则来计算权值和偏差的变化，并使网络误差的平方和最小化，总能够训练一个网络的误差趋于这个最小值。另外很显然， $E(W,B)$只取决于网络的权值及目标矢量。我们的目的是通过调节权矢量，使 $E(W,B)$达到最小值。所以在给定 $E(W,B)$后，利用 W-H 学习规则修正权矢量和偏差矢量，使 $E(W,B)$从误差空间的某一点开始，沿着 $E(W,B)$的斜面向下滑行。根据梯度下降法，权矢量的修正值正比于当前位置上 $E(W,B)$的梯度，对于第 i 个输出节点有:

$$\Delta w_{ij}=-\eta\frac{\partial E}{\partial w_{ij}}=\eta[t_i-a_i]p_j \tag{3.2}$$

或表示为:

$$\Delta w_{ij}=\eta\,\delta_i\,p_j \tag{3.3}$$
$$\Delta b_i=\eta\,\delta_i$$

这里 δ_i 定义为第 i 个输出节点的误差:

$$\delta_i=t_i-a_i \tag{3.4}$$

(3.3)式称为 W-H 学习规则，又叫 δ 规则，或为最小均方差算法(LMS)。 W-H 学习规则的权值变化量正比于网络的输出误差及网络的输入矢量。它不需求导数，所以算法简单，又具有收敛速度快和精度高的优点。

(3.3)式中的 η 为学习速率。在一般的实际运用中，实践表明， η 通常取一接近 1 的数，或取值为:

$$\eta=0.99*\frac{1}{\max[\det(P*P^T)]} \tag{3.5}$$

这样的选择可以达到既快速又正确的结果。

学习速率的这一取法在神经网络工具箱中用函数 **maxlinlr.m** 来实现。(3.5)式可实现为:

```
lr = 0.99 * maxlinlr ( P,1 );
```

其中 lr 为学习速率。

W-H 学习规则的函数为: **learnwh.m** 来实现，另外，加上线性自适应网络输出函数 **purelin.m** ，可以写出 W-H 学习规则的计算公式为:

```
A = purelin ( W*P );
E = T - A;
[ dW, dB ] = learnwh ( P, E, lr );
W = W + dW;
B = B + dB;
```

采用 W-H 规则训练自适应线性元件使其能够得以收敛的必要条件是被训练的输入矢量必须是线性独立的，且应适当地选择学习速率以防止产生振荡现象。

3.3 网络训练

自适应线性元件的网络训练过程可以归纳为以下三个步骤:

1）表达: 计算训练的输出矢量 $A = W*P + B$，以及与期望输出之间的误差 $E = T - A$；

2）检查: 将网络输出误差的平方和与期望误差相比较，如果其值小于期望误差，或训练已达到事先设定的最大训练次数，则停止训练; 否则继续;

3）学习: 采用 W-H 学习规则计算新的权值和偏差，并返回到 1）。

每进行一次上述三个步骤，被认为是完成一个训练循环次数。

如果网络训练获得成功，那么当一个不在训练中的输入矢量输入到网络中时，网络趋于产生一个与其相联想的输出矢量。这个特性被称为泛化，这在函数逼近以及输入矢量分类的应用中是相当有用的。

如果经过训练，网络仍不能达到期望目标，可以有两种选择: 或检查一下所要解决的问题，是否适用于线性网络; 或对网络进行进一步的训练。

虽然只适用于线性网络， W-H 学习规则仍然是重要的，因为它展现了梯度下降法是如何来训练一个网络的，此概念后来发展成反向传播法，使之可以训练多层非线性网络。

采用 MATLAB 进行自适应线性元件网络的训练过程如下:

```
% 表达式
A = purelin ( W*P, B );
E = T - A;
SSE = sumsqr ( E );               % 求误差平方和
for epoch = 1 : max_epoch         % 循环训练
  if SEE   < err_goal             % 比较误差
    epoch = epoch - 1;
    break                         % 若满足期望误差要求，结束训练
  end
[ dW, dB ] = learnwh ( P, E, lr );  % 修正权值
W = W + dW;
```

$$W = -0.2354; \quad B = 0.7066$$

实际上，对于【例 3.1 】这个简单的例题，它存在一个精确解，且可以用解二元一次方程的方式将 P 和 T 值分别对应地代入方程 $T = W*P + B$ 得:

$$\begin{cases} W + B = 0.5 \\ -1.2W + B = 1.0 \end{cases}$$

可解出 $e = T - A = 0$ 的解为:

$$W = -0.2273; B = 0.7273$$

由此看出，对于特别简单的问题，采用自适应线性网络的训练不一定能够得到足够精确的解。因为当训练误差达到期望误差值后，训练即被终止。

对于具有零误差的自适应线性网络，即输入/输出矢量对存在着严格的线性关系，此时的自适应线性网络的设计可以采用工具箱中另外一个名为 **solvelin.m** 的函数。以【例 3.1 】为例，可简单用下面的命令:

```
》[ W, B ] = solvelin ( P, T )
```

可立即得到精确解:

```
W = -0.2273
B = 0.7273
```

然后可用 **simulin.m** 函数来检测所设计的网络:

```
》 A = simulin ( P, W, B )
A =
    0.5000    1.0000
```

还可以用 **sumsqr.m** 函数来求出误差平方和:

```
》 SSE = sumsqr ( T - A )
SSE =
     0
```

可见所设计网络的误差为零。

【例 3.2 】现在来考虑一个较大的多神经元网络的模式联想的设计问题。输入矢量和目标矢量分别为:

$$P = [\,1 \quad 1.5 \quad 1.2 \quad -0.3\,; \quad -1 \quad 2 \quad 3 \quad -0.5\,; \quad 2 \quad 1 \quad -1.6 \quad 0.9\,];$$

$$T = [\,0.5 \quad 3 \quad -2.2 \quad 1.4\,; \quad 1.1 \quad -1.2 \quad 1.7 \quad -0.4\,; \quad 3 \quad 0.2 \quad -1.8 \quad -0.4\,; \quad -1 \quad 0.1 \quad -1.0 \quad 0.6\,];$$

解:

由输入矢量和目标输出矢量可得: $r = 3, s = 4, q = 4$。所以网络的结构如图 3.2 所示。

这个问题的求解同样可以采用线性方程组求出，即对每一个输出节点写出输入和输出之间的关系等式。对于网络每一个输出神经元都有 4 个变量（三个权值加一个偏差），有

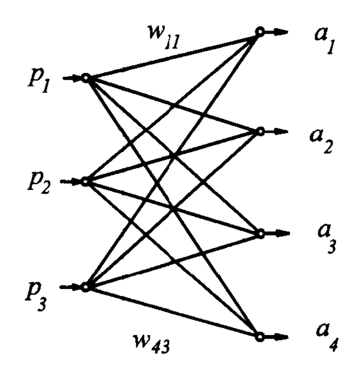


图 3.2　网络设计的结构图

四个限制方程（每组输入对应四个输出值）。这样由四个输出节点，共产生 16 个方程，其方程数目与权值数目相等。所以只要输入矢量是线性独立的，则同时满足这 16 个方程的权值存在且唯一。对应到网络上来说，则存在零误差的精确权值。

实际上要求出这 16 个方程的解是需要花费一定的时间的，甚至是不太容易的。对于一些实际问题，常常并不需要求出其完美的零误差时的解。也就是说允许存在一定的误差。在这种情况下，采用自适应线性网络求解就显示出它的优越性：因为它可以很快地训练出满足一定要求的网络权值。对于有完美解的网络设计，通过对期望误差平方和的选定，也能通过加长训练次数来获得。

下面给出【例 3.2】的设计程序。为了便于对比，在程序中增加了以下几项：

1）训练前由初始权值所得到的误差平方和；

2）最终误差及训练次数的显示；

3）整个训练过程中的网络误差的记录显示；

4）最终权值。

```
% wf2.m
%
P = [ 1 1.5 1.2 -0.3 ; -1 2 3 -0.5 ;2 1 -1.6 0.9 ] ;
T = [0.5 3 -2.2 1.4 ; 1.1 -1.2 1.7 -0.4 ;3 0.2 -1.8 -0.4; -1 0.1 -1.0 0.6];
disp_freq = 400;                                        % 中间不显示结果
max_epoch = 400;
err_goal = 0.001;
lr = 0.9*maxlinlr (P);
W = [ 1.9978 -0.5959 -0.3517; 1.5543 0.05331 1.3660;     % 初始权值
      1.0672 0.3645 -0.9227; -0.7747 1.3839 -0.3384];
B = [ 0.0746;-0.0642;-0.4256;-0.6433];
SSE = sumsqr ( T - purelin ( W*P, B ));                 % 未训练前误差
fprintf ('Before training, sum squared error = %g.\n', SSE)
% 训练网络
flops(0)
tp = [ disp_freq   max_epoch   err_goal   lr ];         % 设置参数变量 tp
[ W, B , epochs, errors ] = trainwh ( W, B, P, T, tp ); % 进行线性网络权值训练
W                                                       % 显示最终训练权矢量
B                                                       % 显示最终训练偏差矢量
SSE =   sumsqr ( T - purelin ( W*P, B ));               % 最终误差
% 显示结果并给出结论
ploterr (errors),
```

```
fprintf ('\n After % .0f epochs, sum squared error = % g.\n\n', SSE),
fprintf ('Training took %.0f flops.\n',flops),
fprintf ('Trained network operates:' );
  if SSE < err_goal
    disp ('Adequately.' )
  else
    disp ('Inadequately.' )
  end
end
```

运行 wf2.m 在训练 274 次循环后其误差平方和达到 0.000993245 ，而初始误差为 144.463 。在整个训练过程中误差的变化走向如图 3.3 所示。

训练后的网络权值为:

```
W = [ -.2.4612   2. 2843   3.1529;
       2.1895   -1.8132   -2.0605;
       2.0821   -1.2604   0.0480;
      -.16615   0.9779    0.9928];
B = [ -1.0440;1.2099;-0.4448;-0.3137 ];
```

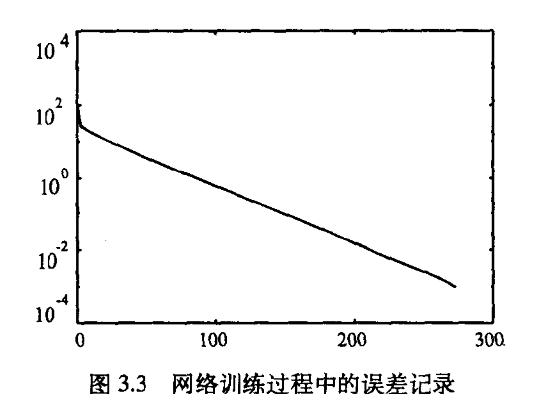


图 3.3　网络训练过程中的误差记录

对于存在零误差的精确权值网络，若用函数 **solvelin.m** 来求解，则更加简单如下:

```
% wf3.m
%
P = [ 1 1.5 1.2 -0.3 ; -1 2 3 -0.5 ;2 1 -1.6 0.9 ] ;
T = [0.5 3 -2.2 1.4 ; 1.1 -1.2 1.7 -0.4 ;3 0.2 -1.8 -0.4; -1 0.1 -1.0 0.6];
[ W, B ] = solvelin ( P, T );
A = simulin ( P, W, B );
SSE = sumsqr ( T - A )
W
B
end
```

由此可得零误差的唯一精确解为:

```
W = [ -2.4914   2.3068   3.1747;
       2.2049  -1.8247  -2.0716;
       2.0938  -1.2691   0.0395;
      -1.6963   0.9815   0.9963];
B = [-1.0512;1.2136;-0.4420;-0.3148]
```

通常可以直接地判断出一个线性网络是否有完美的零误差的解：如果每个神经元所具有的自由度（即权值与阈值数）等于或大于限制数（即输入/输出矢量对），那么线性网络则可以零误差的解决问题。不过这一事实在当输入矢量是线性相关或没有阈值时不成立。

为了演示可能在对线性相关输入矢量的网络训练中出现的问题，现给出下面的例子。

【例 3.3】设计训练一个线性网络实现下列从输入矢量到目标矢量的变换：

$$P = [1 \quad 2 \quad 3;\quad 4 \quad 5 \quad 6];$$

$$T = [0.5\ 1\ -1];$$

由所给输入/输出矢量对可知，所要设计的线性网络应具有阈值，并有两个输入神经元和一个输出神经元。对于这个具有三个可调参数的线性网络，本应得到完美的零误差的精确解。但是不幸的是所给出的输入矢量元素之间是线性相关的：第三组元素等于第二组元素的两倍减去第一组：$p_3 = 2p_2 - p_1$。此时，由于输入矢量的奇异性，用函数 **solvelin.m** 来设计时网络会产生问题。只有在能够线性地解出问题的情况下，用函数 **solvelin.m** 才比较准确。若没有准确的零误差，**solvelin.m** 函数会给出最小误差平方和意义下的解。另外当出现奇异矩阵时，使用 **solvelin.m** 求解权值可能会出现下列警告：

Warning : Matrix is singular to working precision.

而当采用 **trainwh.m** 函数则能训练出达到所给定的训练次数时，或所给定的最小误差意义下的权值时的最小误差。只要将前面已编写的 wf2.m 程序中的输入与目标矢量改变一下，并给出(-1, 1)之间的随机初始值，即可运行看到本例的结果。训练 400 次过程中的误差记录如图 3.4 所示，其最终误差在 1.04 左右，这就是本例题下的最小误差平方和，而当采用完全线性函数的设计 **solvelin.m** 去求解网络权值时，所得到的误差是 4.25。采用 W-H 算法训练出的误差是它的 1/4，由此可见其算法的优越性。

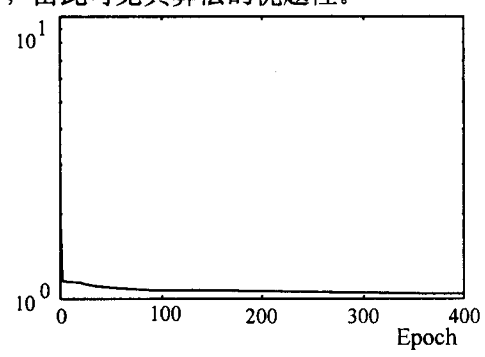


图 3.4 训练过程中的误差记录

【例 3.4】现在假定在【例 3.1】的输入/输出矢量中增加两组元素，使其变为：

$$P = [1.0 \quad 1.5 \quad 3.0 \quad -1.2]$$

$$T = [\,0.5 \quad 1.1 \quad 3.0 \quad -1.0\,]$$

本例题的目的是在于了解自适应线性网络的线性逼近求解的能力。

此时，所要解决的问题相当于用四个方程解 w 和 b 两个未知数。因为方程数大于未知数，所以采用代数方程是无解的。但仍可以用自适应线性网络来对此问题求解。仍然是单个输入节点和单个输出节点，网络具有一个权值 w 和一个偏差 b，通过选定期望误差平方和的值和最大循环次数后，可以让网络进行自行训练和学习。

图 3.5 给出了输入/输出对的位置以及网络求解的结果。对于所设置的 err_goal = 0.001，在循环训练了 50 次后所得的误差平方和仍然为： SSE = 0.289 。这个值即是本题所能达到的最小误差平方和的值，也是用线性网络所能达到的最佳结果。当采用线性自适应线性网络求解问题所得到的误差特别大时，可以认为此问题不适宜用线性网络来解决。

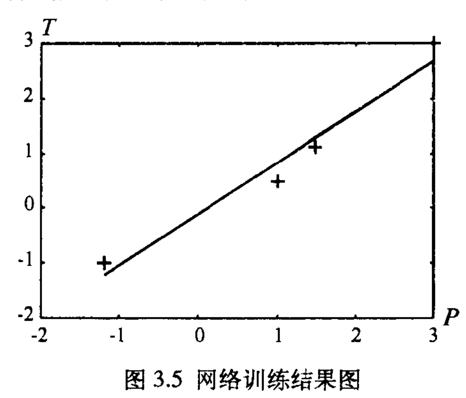


图 3.5 网络训练结果图

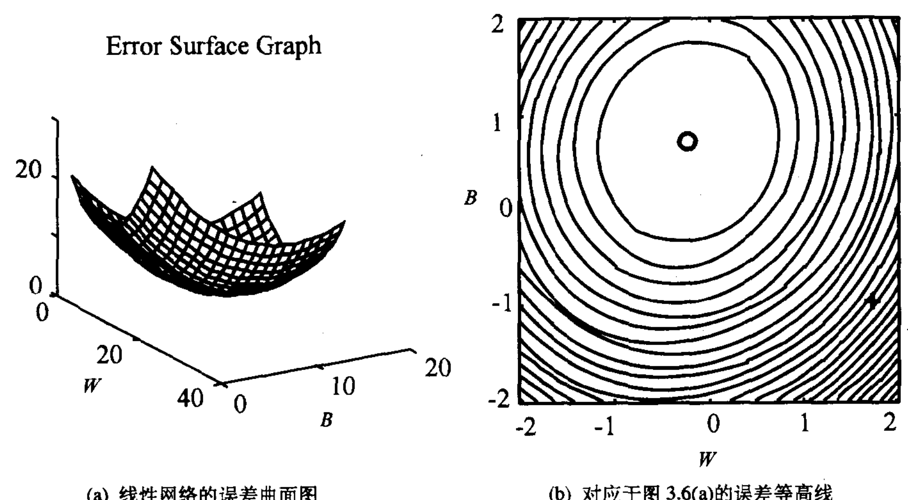
由上面的例子可以看出，自适应线性网络仅可以学习输入/输出矢量之间的线性关系，所以对某些问题，自适应线性网络是不能得到满意的结果的。然而，当一个完美的结果不存在，但只要选择足够小的学习速率，自适应线性网络将总是可以使其误差平方和为最小，网络对其给定的结构，总可得到一个尽可能接近目标的结果。这是因为一个线性网络的误差表面是抛物线型的，既然抛物线只有唯一最小值，基于梯度下降法的 W-H 规则肯定在其最小值处产生结果。

自适应线性网络还有另一个潜在的困难，当学习速率取得较大时，可导致训练过程的不稳定。

【例 3.5 】在网络设计训练中，学习速率的选择是影响收敛速度甚至结果的一个很重要的因素。只要学习速率足够小，采用 W-H 规则总可以训练出一个使其输出误差为最小的线性网络。这里展现一个采用大于所建议的 **maxlinlr.m** 学习速率进行训练情况的例子。

输入/目标矢量与【例 3.1 】相同。我们将以不同的学习速率训练两次网络以展现两种不希望的学习速率带来的影响。

为了能够清楚地观察到网络训练过程中权值的修正所带来的误差的变化情况，我们在程序中加入误差等高线图，并在其中显示每一次权值修正后的误差值位置，如图 3.6 所示。其中图 3.6(a)为由权值 w 和偏差 b 所决定的线性网络误差曲面，可以看到它是一个抛物线型的。图 3.6(b)为图 3.6(a)的从上往下的投影图，图中的曲线称为等高线，线上的点具有相同的误差值。图中的记号“o”代表用函数 **solvelin.m** 求出的误差最小值；而记号“+”为训练前的初始值点。



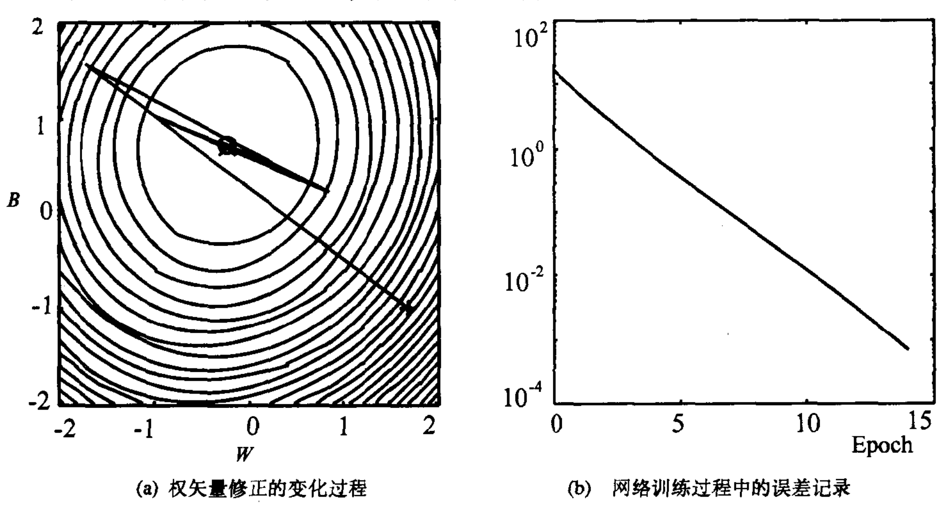
(a) 线性网络的误差曲面图　　(b) 对应于图 3.6(a)的误差等高线

图 3.6　线性网络的误差图形

1 ）对于第一个尝试，学习速率 lr 取:

lr = 1.7*maxlinlr (P);

图 3.7 给出了网络在此学习速率下的权值训练及其误差的记录。



(a) 权矢量修正的变化过程　　(b) 网络训练过程中的误差记录

图 3.7　lr = 1.7*maxlinlr (P)时的网络训练过程

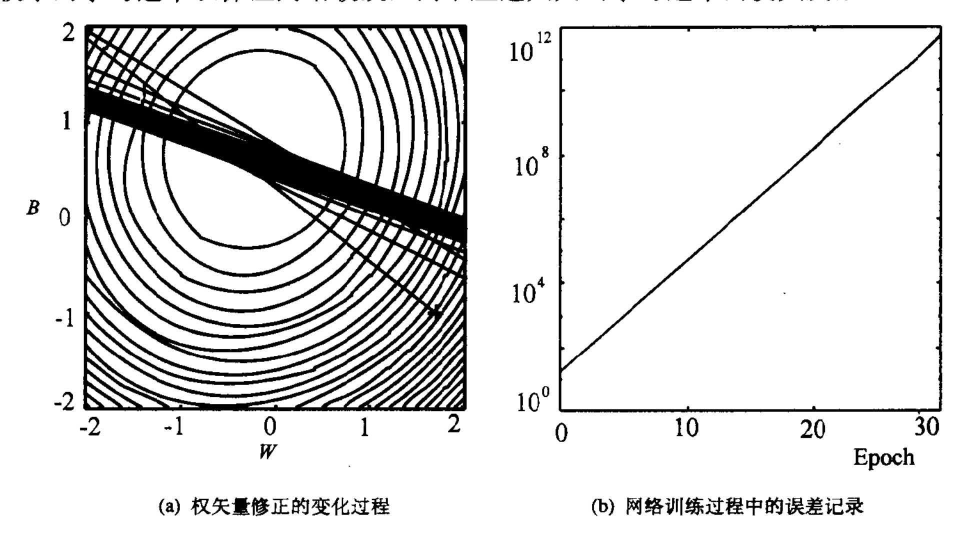
正如所看到的，由于学习速率太大，虽然能够使权值得到较大的修正，但由于过量而产生了振荡。不过误差在其权值振荡过程中仍然能够直线下降，所以在经过 14 次循环后（注意在【例 3.1 】中只用了 12 次）达到了误差的最小值。如果学习速率再加大，可能就不是这么幸运了。

2 ）第二个尝试是选用更大学习速率:

lr = 2.5*maxlinlr (P);

图 3.8 同样给出了网络的权值训练及其误差的记录。从中可以看出，由于其学习速率太大，网络的权值修正过程总是在最小误差方向上运动，但每一次都由于过大的调整使其

偏离期望目标越来越远，其误差是向增加而不是减少的方向移动。由此可以得出结论：应选取较小的学习速率以保证网络收敛，而不应选太大的学习速率而使其发散。



(a) 权矢量修正的变化过程 (b) 网络训练过程中的误差记录

图 3.8 lr = 2.5*maxlinlr (P)时的网络训练过程

最后给出用 MATLAB 工具箱编写本例题的程序:

```
% wf4.m
%
P = [ 1 -1.2];
T = [ 0.5 1 ];
[ R, Q ] = size (P);
[ S, Q ] =size (T);
[ W, B ] = rands ( S, R );
% 画误差曲面图
Wrange = -2:0.2:2; Brang = -2:0.2:2;
ES = errsurf ( P, T, Wrange, Brange, 'purelin');
mesh ( ES, [60 30]);
title ('Error Surface Graph')
pause
% 设计网络权值并画出投影图
[ DW, DB ] = solvelin ( P, T );              % 求出理想网络权值
contour ( ES, 25, Wrange, Brange );          % 画误差的等高线
axis ('equal')
hold on
plot ( DW, DB, 'oy')
xlabel ('W'), ylabel ('B')
pause
```

```
% 选择学习速率
LR = menu ('Use a learning rate of: ', ...
      '1.7 * maxlinlr', ...
      '2.5 * maxlinlr');
disp ('')
%训练权值
disp_freq = 1; max_epoch = 30;   err_goal = 0.001;
if LR == 1
  lr = 1.7*maxlinlr ( P, 1 );
else
  lr = 2.5*maxlinlr ( P, 1 );
  end
flops (0)
pausetime = 1;                                          % 设置暂停时间为 1 秒钟
A = purelin ( W*P, B );
E = T - A;     SSE = sumsqr ( E );
errors = [ SEE ];                                       % 误差记录
for epoch = 1 : max_epoch
if SSE < err_goal, epoch = epoch - 1; break, end
LW = W; LB = B;
[ dW, dB ] = learnwh ( P, E, lr );
W = W + dW; B = B + dB;
A = purelin ( W*P, B );  E = T - A;     SSE = sumsqr ( E );   % 学习后的再次表达式
errors = [ errors SEE ];                                % 误差的再次记录
% 训练过程中的显示
  if rem ( epoch, disp_freq ) == 0
  temp = flops;
  plot ([LW W],[LB B],'r-');
  drawnow
  flops (temp);
  pause2(pausetime)
  end
end
%显示误差记录
A = purelin ( W*P, B );
ploterr (errors)
end
```

3.5 对比与分析

到目前为止，我们已经学习了感知器和自适应线性网络。这两类网络在其结构和学习算法上都没有什么太大的差别，甚至是大同小异。我们前面在人工神经网络的发展历史中已经介绍过，这两种模型实际上是最早的模型，且自适应线性网络是在感知器的研究基础上建立发展起来的。不过就是从它们细小的区别上，我们也能够看到它们之间功能的不同点，而这些不同点，在其它各种神经网络中表现得更加突出。所以在此想再次强调一下学习人工神经网络的重点在于掌握不同的概念上，在于掌握不同网络名称的不同特点上。这些特点主要表现在下面三点上：

（1）网络模型结构上

一种网络有一种结构。我们已学过的感知器和自适应线性网络，几乎具有相同的结构，但其他网络如霍普菲尔德网络就完全是另外一种结构。可以说所有的不同的网络都是为了完成某一需要而设计成特有的网络模型结构。我们必须了解各自独特的结构以及所达到的不同作用。只有这样，我们才能够设计出符合特殊需要的其他网络结构来。

就感知器和自适应线性网络而言，结构上的主要区别在于激活函数：一个是二值型的，一个线性的。仅此一点，就使得感知器仅能做简单的分类工作，而自适应线性网络除了有分类功能外，还可以进行线性逼近。当把偏差与权值考虑成一体时，自适应线性网络的输入与输出之间的关系可以写成 $A = W*P$ 。如果 P 是满秩的话，则可以写成 $AP^{-1}=W$ ，或 $W=\frac{A}{P}$。从自动控制理论的角度来说， W 就是系统的传递函数，只不过 W 不是计算出来的，而是训练出来的，且自适应线性网络中的 W 只能表现出输入与输出之间的线性关系。当输入与输出之间是非线性关系时，通过对网络的训练，可以得出线性逼近关系，即线性化的传递函数。这个思想可以运用在对被控系统的建模上，即系统辨识上。当然，若把此思想用在后面将要学习到的 BP 网络中，则意味着可以对任意非线性被控系统建立精确的系统模型。这无论用自动控制的经典理论，还是用现代控制理论，都是无法做到的，因为在那里，只能对线性的或线性化的系统建立传递函数。而运用人工神经网络，采用最简单的自适应线性网络即可做到。

（2）学习算法

学习算法是为了完成不同的任务、达到不同的目的所建立的，所以不同的模型，有不同的算法。感知器的算法是最早提出的可收敛的算法，它的自适应思想被威德罗和霍夫发展成使其误差最小的梯度下降法。最后又在 BP 算法中得到进一步的推广，它们属于同一类算法。除此以外，在其它模型中，则采用各种完全不同的算法。后面，我们还将学习到的算法有：海布学习算法、内星法、外星法和科荷伦法等。

（3）适用性与局限性

每个网络模型以其独特的结构和学习算法，决定了它所能够解决的问题，这就给出了它的适用性，同时也指出了它的局限性。比如，感知器仅能够进行简单的分类。从前面的例题中已经看出，感知器可以将输入分成两类或四类等。它的局限性是仅能对线性可分的

输人进行分类。遗憾的是，哪些输人是线性可分的，在划分前我们并没有办法得知，只有进行训练后才能知道是否线性可分。不过理论已经证明，对于线性可分的输人矢量，感知器在有限的时间里一定可以达到结果。

自适应线性网络除了像感知器一样可以进行线性分类外，又多了线性逼近，这仅是由于其激活函数可以连续取值而不同于感知器的仅能取 0 或 1 的缘故。后面将要学习到的 BP 网络又以不同的网络结构以及多层学习算法，而能够进行诸如非线性函数逼近与压缩等其它模型所没有的功能，但它也有自身的缺点与局限性，如：需要的训练时间较长，易陷人局部最小值等。再如，霍普菲尔德网络以完全另一种网络结构和学习算法能够达到自己收敛到自己的功能，但实际上也存在着很难找到这种收敛的情况。

总而言之，我们应从以上三个方面去学习人工神经网络。只有这样，当我们自己遇到一个实际问题时，才能够正确选择进而达到设计出有效的神经网络来。

3.6 本章小结

1 ）自适应线性网络仅可以学习输入/输出矢量之间的线性关系，可用于模式联想及函数的线性逼近。网络结构的设计完全由所要解决的问题所限制，网络的输入数目和输出层中神经元数目，由问题所限制;

2 ）多层线性网络不产生更强大的功能，从这个观点上看，单层线性网络不比多层线性网络有局限性;

3 ）输入和输出之间的非线性关系不能用一个线性网络精确地设计出，但线性网络可以产生一个具有误差平方和最小的线性逼近。

习 题

〖3.1〗试用 MATLAB 编写对【例 3.4】进行网络设计的程序，使期望误差和 err_goal = 0.001 ，并观察训练次数达到 50 次后的网络误差结果，判断是否适用于用自适应线性网络来解决此问题。

〖3.2〗设计一个具有单元输入和单元输出的线性网络，注意观察其解的特性:

$$P = 1, T = 0.5$$

第四章　反向传播网络

反向传播网络（Back-Propagation Network，简称BP网络）是将W-H学习规则一般化，对非线性可微分函数进行权值训练的多层网络。

BP网络主要用于:

1）函数逼近: 用输人矢量和相应的输出矢量训练一个网络逼近一个函数;

2）模式识别: 用一个特定的输出矢量将它与输入矢量联系起来;

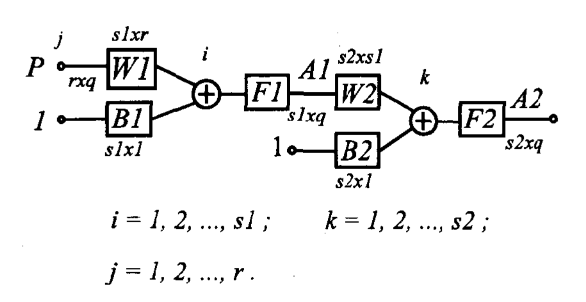
3）分类: 把输人矢量以所定义的合适方式进行分类;

4）数据压缩: 减少输出矢量维数以便于传输或存储。

在人工神经网络的实际应用中，80%~90%的人工神经网络模型是采用BP网络或它的变化形式，它也是前向网络的核心部分，体现了人工神经网络最精华的部分。在人们掌握反向传播网络的设计之前，感知器和自适应线性元件都只能适用于对单层网络模型的训练，只是后来才得到了进一步拓展。

4.1　BP网络模型与结构

一个具有 r 个输入和一个隐含层的神经网络模型结构如图4.1所示。



$i = 1, 2, \ldots, s1$;　　$k = 1, 2, \ldots, s2$;

$j = 1, 2, \ldots, r$.

图4.1　具有一个隐含层的神经网络模型结构图

感知器和自适应线性元件的主要差别在激活函数上: 前者是二值型的，后者是线性的。BP网络具有一层或多层隐含层，除了在多层网络上与前面已介绍过的模型有不同外，其主要差别也表现在激活函数上。BP网络的激活函数必须是处处可微的，所以它就不能采用二值型的阀值函数{0, 1}或符号函数{-1, 1}，BP网络经常使用的是S型的对数或正切激活函数和线性函数。

图4.2所示的是S型激活函数的图形。可以看到 $f(\cdot)$ 是一个连续可微的函数，其一阶导数存在。对于多层网络，这种激活函数所划分的区域不再是线性划分，而是由一个非线性的超平面组成的区域。它是比较柔和、光滑的任意界面，因而它的分类比线性划分精确、合理，这种网络的容错性较好。另外一个重要的特点是由于激活函数是连续可微的，它可

以严格利用梯度法进行推算，它的权值修正的解析式十分明确，其算法被称为误差反向传播法，也简称 BP 算法，这种网络也称为 BP 网络。

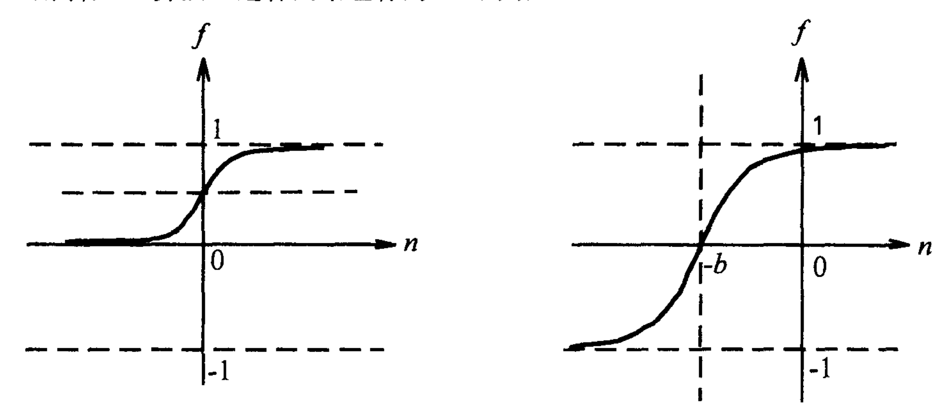


图 4.2 BP 网络 S 型激活函数

因为 S 型函数具有非线性放大系数功能，它可以把输入从负无穷大到正无穷大的信号，变换成-1 到 1 之间输出，对较大的输入信号，放大系数较小；而对较小的输入信号，放大系数则较大，所以采用 S 型激活函数可以去处理和逼近非线性的输入/输出关系。不过，如果在输出层采用 S 型函数，输出则被限制到一个很小的范围了，若采用线性激活函数，则可使网络输出任何值。所以只有当希望对网络的输出进行限制，如限制在 0 和 1 之间，那么在输出层应当包含 S 型激活函数，在一般情况下，均是在隐含层采用 S 型激活函数，而输出层采用线性激活函数。

4.2 BP 学习规则

BP 网络的产生归功于 BP 算法的获得。BP 算法属于δ 算法，是一种监督式的学习算法。其主要思想为：对于 q 个输入学习样本：$P^1, P^2, ..., P^q$，已知与其对应的输出样本为：$T^1, T^2, ..., T^q$。学习的目的是用网络的实际输出 $A^1, A^2, ..., A^q$ 与目标矢量 $T^1, T^2, ..., T^q$ 之间的误差来修改其权值，使 A^l，($l = 1, 2, ..., q$) 与期望的 T^l 尽可能地接近；即：使网络输出层的误差平方和达到最小。它是通过连续不断地在相对于误差函数斜率下降的方向上计算网络权值和偏差的变化而逐渐逼近目标的。每一次权值和偏差的变化都与网络误差的影响成正比，并以反向传播的方式传递到每一层的。

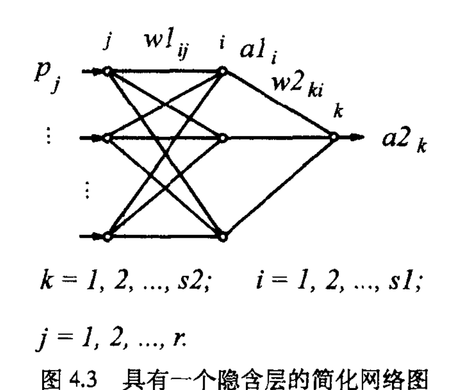


图 4.3 具有一个隐含层的简化网络图

BP 算法是由两部分组成：信息的正向传递与误差的反向传播。在正向传播过程中，输入信息从输入经隐含层逐层计算传向输出层，每一层神经元的状态只影响下一层神经元的状态。如果在输出层没有得到期望的输出，则计算输出层的误差变化值，然后转向反

向传播，通过网络将误差信号沿原来的连接通路反传回来修改各层神经元的权值直至达到期望目标。

为了明确起见，现以图 4.1 所示两层网络为例进行 BP 算法推导，其简化图如图 4.3 所示。

设输入为 P，输入神经元有 r 个，隐含层内有 $s1$ 个神经元，激活函数为 $F1$，输出层内有 $s2$ 个神经元，对应的激活函数为 $F2$，输出为 A，目标矢量为 T。

4.2.1 信息的正向传递

1）隐含层中第 i 个神经元的输出为：

$$a1_i = f1\left(\sum_{j=1}^{r} w1_{ij} p_j + b1_i\right),\ i = 1, 2, \ldots, s1 \tag{4.1}$$

2）输出层第 k 个神经元的输出为：

$$a2_k = f2\left(\sum_{i=1}^{s1} w2_{ki} a1_i + b2_k\right),\ k = 1, 2, \ldots, s2 \tag{4.2}$$

3）定义误差函数为：

$$E(W,B) = \frac{1}{2}\sum_{k=1}^{s2}(t_k - a2_k)^2 \tag{4.3}$$

4.2.2 利用梯度下降法求权值变化及误差的反向传播

（1）输出层的权值变化

对从第 i 个输入到第 k 个输出的权值有：

$$\begin{aligned}\Delta w2_{ki} &= -\eta\frac{\partial E}{\partial w2_{ki}} = -\eta\frac{\partial E}{\partial a2_k}\frac{\partial a2_k}{\partial w2_{ki}} \\ &= \eta(t_k - a2_k)\ f2'\quad a1_i = \eta\ \delta_{ki}\ a1_i\end{aligned} \tag{4.4}$$

其中：

$$\delta_{ki} = (t_k - a2_k) = e_k\ f2' \tag{4.5}$$

$$e_k = t_k - a2_k \tag{4.6}$$

同理可得：

$$\begin{aligned}\Delta b2_{ki} &= -\eta\frac{\partial E}{\partial b2_{ki}} = -\eta\frac{\partial E}{\partial a2_k}\cdot\frac{\partial a2_k}{\partial b2_{ki}} \\ &= \eta(t_k - a2_k)\cdot f2' = \eta\cdot\delta_{ki}\end{aligned} \tag{4.7}$$

（2）隐含层权值变化

对从第 j 个输入到第 i 个输出的权值，有：

$$\begin{aligned}\Delta w1_{ij} &= -\eta\frac{\partial E}{\partial w1_{ij}} = -\eta\frac{\partial E}{\partial a2_k}\cdot\frac{\partial a2_k}{\partial a1_i}\cdot\frac{\partial a1_i}{\partial w1_{ij}} \\ &= \eta\sum_{k=1}^{s2}(t_k - a2_k)\cdot f2'\cdot w2_{ki}\ f1'\cdot p_j = \eta\cdot\delta_{ij}\cdot p_j\end{aligned} \tag{4.8}$$

其中:

$$\delta_{ij} = e_i \cdot f1', \quad e_i = \sum_{k=1}^{s2} \delta_{ki} w2_{ki} \tag{4.9}$$

同理可得:

$$\Delta b1_i = \eta \delta_{ij} \tag{4.10}$$

在 MATLAB 工具箱中，上述公式的计算均已编成函数的形式，通过简单的书写调用即可方便地获得结果。具体有:

1 ）对于(4.1)式所表示的隐含层输出，若采用对数 S 型激活函数，则用函数 **logsig.m**；若采用双曲正切 S 型激活函数，则用函数 **tansig.m**；

2 ）对于(4.2)式所表示的输出层输出，若采用线性激活函数有 **purelin .m** 与之对应;

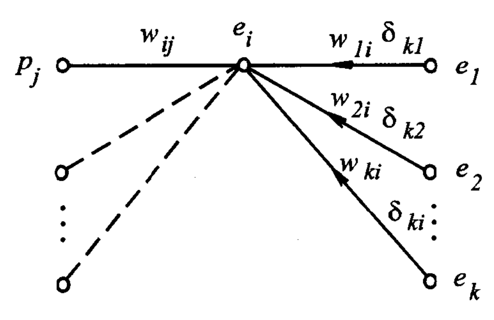
3 ）对于(4.3)式所表示的误差函数，可用函数 **sumsqr.m** 求之;

4 ）有 **learnbp.m** 函数专门求(4.4)、(4.7)、(4.8)和(4.10)式所表示的输出层以及隐含层中权值与偏差的变化量;

5 ）由(4.5)和(4.9)式所表示的误差的变化有函数 **deltalin.m**， **deltatan.m** 和 **deltalog.m** 来解决。它们分别用于线性层、双曲正切层和对数层。

4.2.3 误差反向传播的流程图与图形解释

误差反向传播过程实际上是通过计算输出层的误差 e_k，然后将其与输出层激活函数的一阶导数 $f2'$ 相乘来求得 δ_{ki}。由于隐含层中没有直接给出目标矢量，所以利用输出层的 δ_{ki} 进行误差反向传递来求出隐含层权值的变化量 $\Delta w2_{ki}$。然后计算 $e_i = \sum_{k=1}^{s2} \delta_{ki} \ w2_{ki}$ ，并同样通过将 e_i 与该层激活函数的一阶导数 $f1'$ 相乘，而求得 δ_{ij}，以此求出前层权值的变化量 $\Delta w1_{ij}$。如果前面还有隐含层，沿用上述同样方法依此类推，一直将输出误差 e_k 一层一层的反推算到第一层为止。图 4.4 给出了形象的解释。



$k = 1, 2, ..., s2; \quad i = 1, 2, ..., s1;$

$j = 1, 2, ..., r.$

图 4.4 误差反向传播法的图形解释

BP 算法要用到各层激活函数的一阶导数，所以要求其激活函数处处可微。对于对数 S 型激活函数 $f(n)=\dfrac{1}{1+e^{-n}}$，其导数为：

$$f(n)=\frac{0-e^{-n}(-1)}{(1+e^{-n})^2}=\frac{1}{(1+e^{-n})^2}(1+e^{-n}-1)$$

$$=\frac{1}{1+e^{-n}}(1-\frac{1}{1+e^{-n}})=f(n)[1-f(n)]$$

对于线性函数的导数有：

$$f'(n)=n'=1$$

所以对于具有一个 S 型函数的隐含层，输出层为线性函数的网络，有：

$$f2'=1, \quad f1'=a(1-a)$$

4.3 BP 网络的训练过程

为了训练一个 BP 网络，需要计算网络加权输入矢量以及网络输出和误差矢量，然后求得误差平方和。当所训练矢量的误差平方和小于误差目标，训练则停止，否则在输出层计算误差变化，且采用反向传播学习规则来调整权值，并重复此过程。当网络完成训练后，对网络输入一个不是训练集合中的矢量，网络将以泛化方式给出输出结果。

在动手编写网络的程序设计之前，必须首先根据具体的问题给出的输入矢量 P 与目标矢量 T，并选定所要设计的神经网络的结构，其中包括以下内容：

①网络的层数；②每层的神经元数；③每层的激活函数。

由于 BP 网络的层数较多且每层神经元也较多，加上输入矢量的组数庞大，往往使得采用一般的程序设计出现循环套循环的复杂嵌套程序，从而使得程序编得既费时，又不易调通，浪费了大量的时间在编程中而无暇顾及如何设计出具有更好性能的网络来。在这点上 MATLAB 工具箱充分展示出其神到之处。它的全部运算均采用矩阵形式，使其训练既简单，又明了快速。为了能够较好地掌握 PB 网络的训练过程，下面我们仍用两层网络为例来叙述 BP 网络的训练步骤。

1）用小的随机数对每一层的权值 W 和偏差 B 初始化，以保证网络不被大的加权输入饱和；并进行以下参数的设定或初始化：

a)期望误差最小值 err_goal；

b)最大循环次数 max_epoch；

c)修正权值的学习速率 lr，一般情况下 lr = 0.01 ~ 0.7；

f)从 1 开始的循环训练： for epoch = 1 : max_epoch；

2）计算网络各层输出矢量 A1 和 A2 以及网络误差 E：

```
A1 = tansig ( W1*P, B1);
```

```
A2 = purelin ( W2*A1, B2);
E = T - A;
```

3 ）计算各层反传的误差变化 D2 和 D1 并计算各层权值的修正值以及新权值:

```
D2 = deltalin ( A2, E );
D1 = deltatan ( A1, D2, W2 );
[ dW1, dB1 ] = learnbp ( P, D1, lr );
[ dW2, dB2 ] = learnbp ( A1, D2, lr );
W1 = W1 + dW1; B1 = B1 + dB1;
W2 = W2 + dW2; B2 = B2 + dB2;
```

4 ）再次计算权值修正后误差平方和:

```
SSE = sumsqr(T - purelin(W2*tansig(W1*P,B1),B2));
```

5 ）检查 SSE 是否小于 err_goal,若是，训练结束；否则继续。

以上就是 BP 网络在 MATLAB 中的训练过程。可以看出其程序是相当简单明了的。即使如此，以上所有的学习规则与训练的全过程，仍然可以用函数 **trainbp.m** 来完成。它的使用同样只需要定义有关参数：显示间隔次数，最大循环次数，目标误差，以及学习速率，而调用后返回训练后权值，循环总数和最终误差:

```
TP = [ disp_fqre    max_epoch    err_goal    lr ];
[ W, B, epochs, errors ] = trainbp ( W, B,'F', P, T, TP );
```

函数右端的'F'为网络的激活函数名称。

当网络为两层时，可从第一层开始，顺序写出每一层的权值初始值，激活函数名，最后加上输入、目标输出以及 TP，即:

```
[W1,B1,W2,B2,W3,B3,epochs,errors] = trainbp(W1,B1, 'F1',W2,B2, 'F2',W3,B3, 'F3',P,T,TP);
```

神经网络工具箱中提供了两层和三层的 BP 训练程序，其函数名是相同的，都是 **trainbp.m**，用户可以根据层数来选取不同的参数。

【例 4.1】用于函数逼近的 BP 网络的设计。

一个神经网络最强大的用处之一是在函数逼近上。它可以用在诸如被控对象的模型辨识中，即将过程看成一个黑箱子，通过测量其输入/输出特性，然后利用所得实际过程的输入/输出数据训练一个神经网络，使其输出对输入的响应特性具有与被辨识过程相同的外部特性。

下面给出一个典型的用来进行函数逼近的两层结构的神经网络，它具有一个双曲正切

型的激活函数隐含层，其输出层采用线性函数。

这里有 21 组单输入矢量和相对应的目标矢量，试设计神经网络来实现这对数组的函数关系。

```
P = -1 : 0.1 :1;
T = [ -0.96    0.577    -0.0729   0.377     0.641     0.66     0.461    0.1336 ...
      -0.201   -0.434    -0.5      -0.393    -0.1647   0.0988   0.3072 ...
      0.396    0.3449   0.1816 -0.0312   -0.2183    -0.3201 ];
```

为此，我们选择隐含层神经元为 5 。较复杂的函数需要较多的隐含层神经元，这可以根据试验或经验来确定。

图 4.5 给出了目标矢量相对于输入矢量的图形。一般在测试中常取多于训练用的输入矢量来对所设计网络的响应特性进行测试．在函数逼近的过程中，画出网络输出相对于输入矢量的图形，可以观察到网络在训练过程中的变化过程。为了达到这一目的，我们定义一个密度较大的第二个输入矢量:

P2 = -1 : 0.025 :1;

首先对网络进行初始化:

```
[ R, Q ] = size (P);          [ S2, Q]= size (T);          S1 = 5;
[ W1, B1 ] = rands ( S1, R );
[ W2, B2 ] = rands ( S2, S1 );
```

通过测试，用输入矢量 P2 来计算网络的输出:

A2 = purelin (W2*tansig (W1*P2, B1), B2);

可以画出结果来观察初始网络是如何接近所期望训练的输入/输出关系，如图 4.6 所示。其中，初始网络的输出值用实线给出。

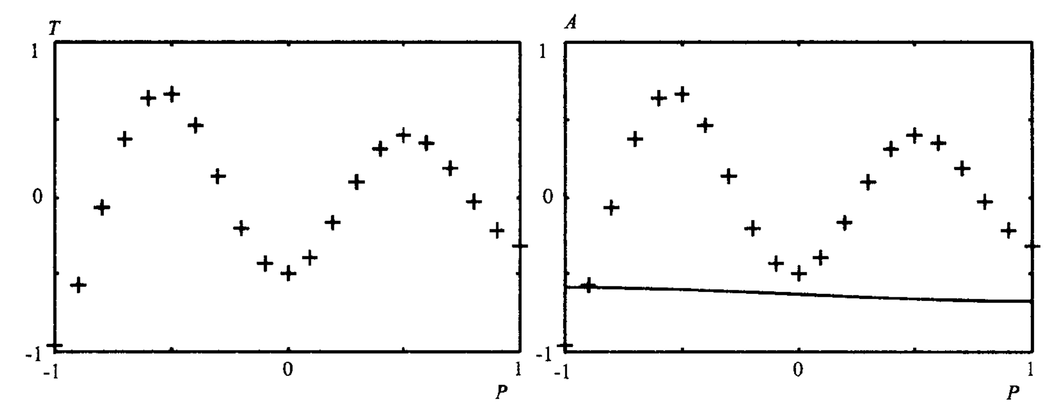


图 4.5　以目标矢量相对于输入矢量的图形　　　　图 4.6　初始网络的输出曲线

网络训练前的误差平方和为 11.9115 ，其初始值为:

```
W10 = [ 0.7771    0.5336    -0.3874    -0.2980    0.0265];
B10 = [0.1822;0.6920;-0.1758;0.6830;-0.4614];
W20 = [-0.1692    0.0746    -0.0642    -0.4256    -0.6433];
B20 = [-0.6926];
```

下面定义训练参数并进行训练:

```
disp_fqre = 10;      max_epoch = 8000; err_goal = 0.02;      lr = 0.01;
TP = [ disp_fqre    max_epoch    err_goal    lr ];
trainbp ( W1, B1, 'tansig', W2 B2, 'purelin', P, T, TP ]
```

由此可以返回训练后的权值、训练次数和偏差。

图 4.7 至图 4.10 给出了网络输出值随训练次数的增加而变化的过程。每个图中标出了循环数目以及当时的误差平方和。

图 4.11 给出了 6801 次循环训练后的最终网络结果，网络的误差平方和落在所设定的 0.02 以内(0.0199968)。

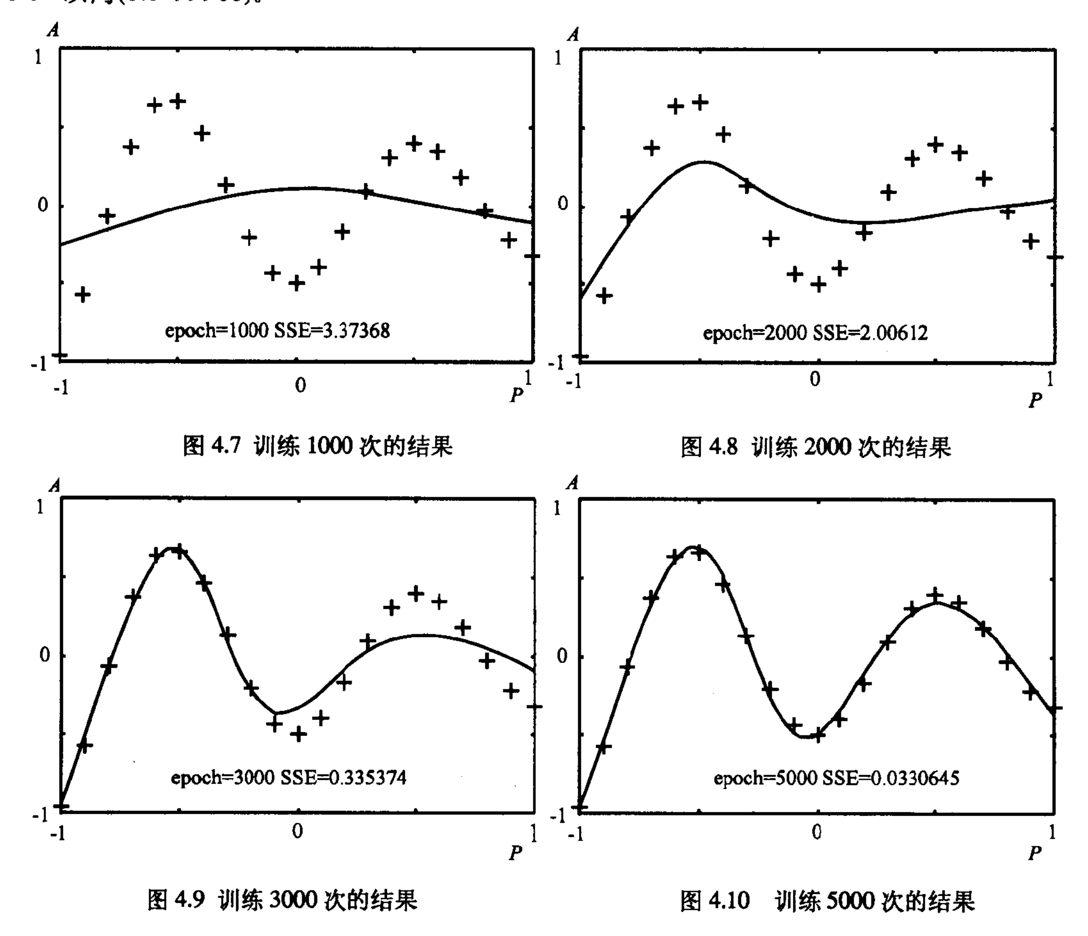


图 4.7 训练 1000 次的结果

图 4.8 训练 2000 次的结果

图 4.9 训练 3000 次的结果

图 4.10 训练 5000 次的结果

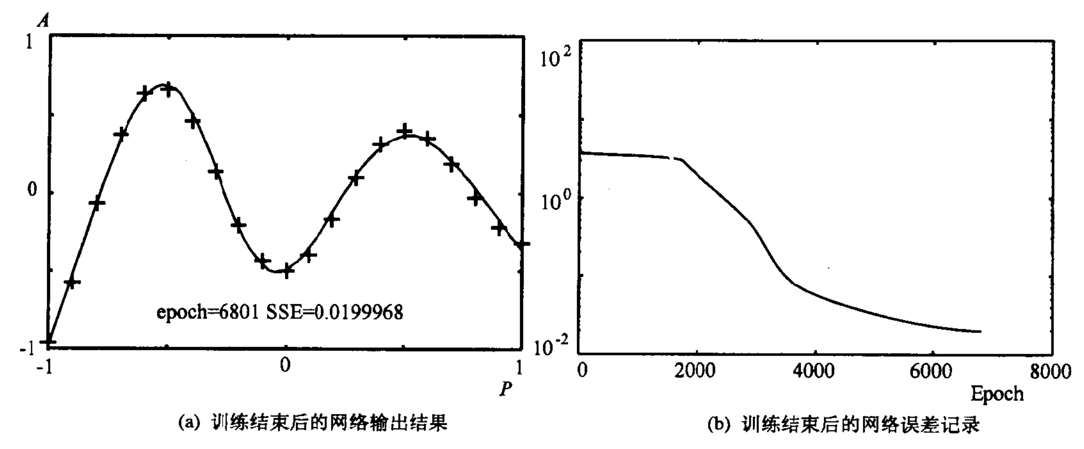


图 4.11 训练结束后的网络输出与误差结果

因为反向传播法采用的是连续可微函数，所以网络对输入/输出的逼近是平滑的。另外，虽然网络仅在输入值-1, -0.9, -0.8, ..., 0.9, 1.0 处进行训练，但对于其他输入值的出现，例如，对训练后的网络输入 $p = 0.33$ 的值，网络的输出端可得其对应输出为：

```
》 A1 = tansig(W1*0.33,B1);
》 A2 = purelin(W2*A1,B2)
A2 =
     0.1659
```

正如所希望的那样，这个值落在输入矢量为 0.3 和 0.4 所对应的输出矢量之间。网络的这个能力使其平滑地学习函数，使网络能够合理地响应被训练以外的输入。这性质称为泛化性能。要注意的是，泛化性能只对被训练的输入/输出对最大值范围内的数据有效，即网络具有内插值特性，不具有外插值性。超出最大训练值的输入必将产生大的输出误差。

4.4 BP 网络的设计

在进行 BP 网络的设计时，一般应从网络的层数、每层中的神经元个数和激活函数、初始值以及学习速率等几个方面来进行考虑。下面讨论一下各自选取的原则。

4.4.1 网络的层数

理论上已经证明：具有偏差和至少一个 S 型隐含层加上一个线性输出层的网络，能够逼近任何有理函数。这实际上已经给了我们一个基本的设计 BP 网络的原则。增加层数主要可以更进一步的降低误差，提高精度，但同时也使网络复杂化，从而增加了网络权值的训练时间。而误差精度的提高实际上也可以通过增加隐含层中的神经元数目来获得，其训

练效果也比增加层数更容易观察和调整。所以一般情况下，应优先考虑增加隐含层中的神经元数。

另外还有一个问题：能不能仅用具有非线性激活函数的单层网络来解决问题呢？结论是：没有必要或效果不好。因为能用单层非线性网络完美解决的问题，用自适应线性网络一定也能解决，而且自适应线性网络的运算速度还更快。而对于只能用非线性函数解决的问题，单层精度又不够高，也只有增加层数才能达到期望的结果。

这主要还是因为一层网络的神经元数被所要解决的问题本身限制造成的。下面给出两个例题来说明此问题。

【例 4.2】考虑两个单元输入的联想问题。其输入和输出矢量分别为:

$$P=[-3 \quad 2], \quad T=[0.4 \quad 0.8]$$

解:

当采用含有一个对数 S 型单层网络求解时，可求得解为:

$$w=0.3350$$
$$b=0.5497$$

此时所达到的误差平方和 $err_goal<0.001$ 。若将这个误差转换成输出误差时，其绝对误差约为 0.02 。

若采用自适应线性网络来实现此联想，得解为:

$$w=0.08$$
$$b=0.64$$

此网络误差为: $e=T-Y=0$ 。

我们还可以从点拟合这个角度来看这个问题。以输入 P 和输出 A 分别为横坐标和纵坐标，可得图 4.12 ，其中，虚线为采用非线性网络拟合的结果。求两点的联想问题实质上是求以此两点所拟合的直线，所以当采用自适应线性网络来求解此问题时，可以得到零误差的完美解。而当采用 S 型函数对此两点进行拟合，因为点数太少，所以反而产生一定的误差。

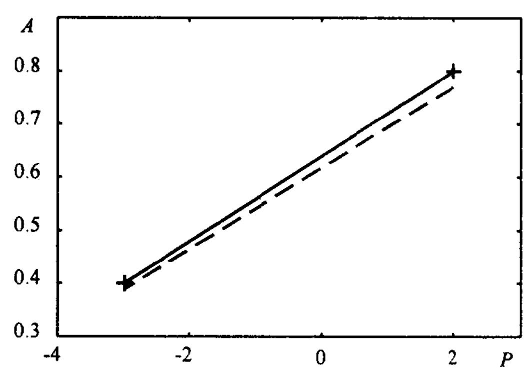


图 4.12　两种不同网络的解

【例 4.3】用一层网络来实现下面的输入/输出关系:

$$P=[-6 \quad -6.1 \quad -4.1 \quad -4 \quad 4 \quad 4.1 \quad 6 \quad 6.1],$$
$$T=[0.0 \quad 0.0 \quad 0.97 \quad 0.99 \quad 0.01 \quad 0.03 \quad 1 \quad 1],$$

解:

这是一个约束大于变量的代数问题，这是因为由输入/输出对可得:

矢量的维数为: $P_{rxq}=P_{1\text{x}8}$, $T_{sxq}=T_{l\text{x}8}$ ，所以约束等式有 8 个。

而权值的维数为: $W_{rxm}=W_{1\text{x}1}$ ，加上一个偏差值 b ，一共只有两个变量。

当采用单层对数非线性网络求解时，求得的一个解为:

$$w = 0.079$$

$$b = -0.1616$$

此时具有误差平方和为 1.85 ，总共进行了 380 次循环修正。

当采用自适当线性网络求解本题时，在具有相同误差平方和时，只用了 40 次循环修正，即得出一解为:

$$w = 0.0204$$

$$b = 0.4537$$

同样可以采用横纵坐标来画出输入和目标输出矢量，如图 4.13 所示。并分别画出两个网络对输入矢量的响应。图中，将输入/目标矢量对用“+”号表示，因为两个网络具有相同的误差，所以在图中几乎看不出它们之间的差别来。

网络的输出反映的是在该网络下所达到的最小误差平方和时，对输入节点的响应。

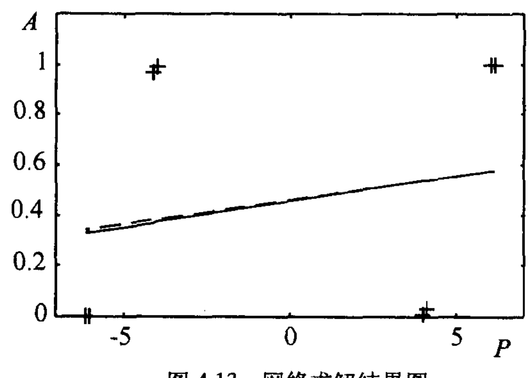


图 4.13　网络求解结果图

很明显本例题的误差不为零，而且非线性与线性网络的响应几乎是完全相同的。这可以使我们得出结论: 既使是采用非线性激活函数，在采用单层网络时，并没有体现出特殊的优越性，甚至还不如线性网络的效果。从图 4.13 中，我们还可以看出，网络的输出直线实际上起到的是一种分类功能。它把四个点分成上下两类。

通过上面两个例题可以看出，对于一般可用一层解决的问题，应当首先考虑用感知器，或自适应线性网络来解决，而不采用非线性网络，因为单层不能发挥出非线性激活函数的特长。

输入神经元数可以根据需要求解的问题和数据所表示的方式来确定。如果输入的是电压波形，那么可根据电压波形的采样点数来决定输入神经元的个数，也可以用一个神经元，使输入样本为采样的时间序列。如果输入为图像，那么则输入可以用图像的象素，也可以为经过处理后的图像特征来确定其神经元个数。总之问题确定后，输入与输出层的神经元数就随之定了。在设计中应当注意尽可能地减少网络模型的规模，以便减少网络的训练时

间。下面我们通过同样简单的单层非线性网络所形成的网络误差曲面，并通过与线性的情况的对比，来进一步了解非线性网络的特性、功能及其优缺点。

【例 4.4】非线性误差曲面特性观察。

理解和掌握反向传播网络工作的最好方式就是观察它是如何工作的。像 W-H 学习规则一样，反向传播法也是企图通过调整每一个正比于误差的变化来修正权值而使误差函数最小化，也是属于梯度下降法。一个简单的比喻就是好像在观察一个没有惯性的弹子（代表网络）在一个粗糙不平的面上滚动。弹子总是在最速方向上滚动直到停留在一个谷的底部。（即误差的极小值）。

为了能够观察非线性误差曲面的形状以及训练时误差的走向，现仍然采用单层非线性网络来求解【例 4.2】，首先观察误差的立体图。它的 MATLAB 编程如下：

```
Wrange = -4 : 0.4 :4;                              % 限制坐标范围
Brange = -4 : 0.4 :4;
ES = errsurf ( P, T, Wrange, Breang, 'logsig' );   % 计算误差曲面
view ( [ 60 30 ] );                                % 显示误差曲面图形
xlabel ('W'),                                      % 写 x 坐标说明
ylabel ('B'),                                      % 写 y 横坐标说明
zlabel ('Sum Squared Error'),                      % 写 z 坐标说明
title ('Error Surface Graph'),                     % 写图标题
```

误差的等高线图用下列命令绘出：

```
contour ( ES, 25, Wrange, Brange);
axis ('equal'),
```

两图分别如图 4.14 和图 4.15 所示，其中图 4.14 中的 z 坐标为误差值。两图中的黑点对应着相同的点。对照两图可以分别看出不同的网络权值所对应的误差值。

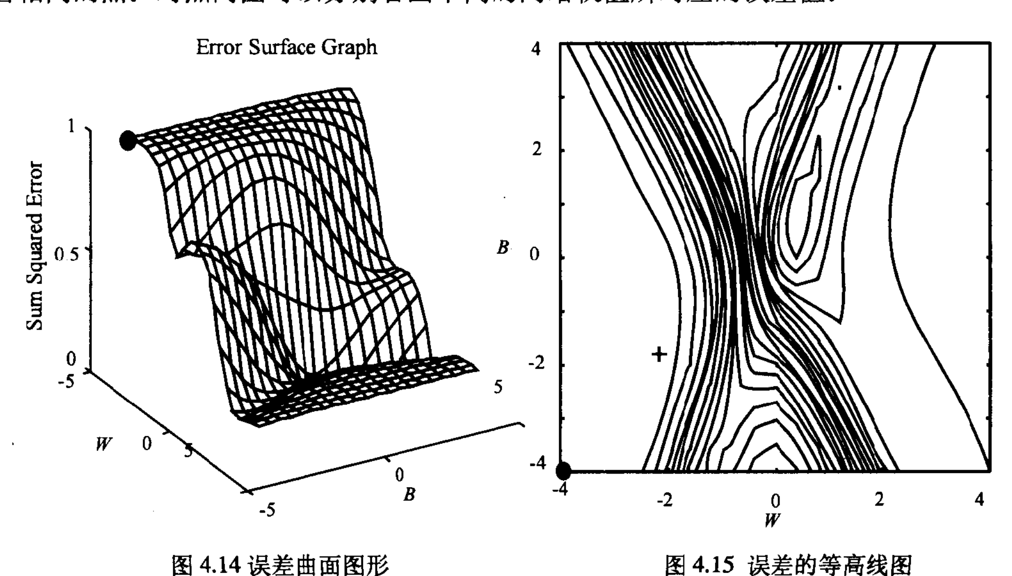
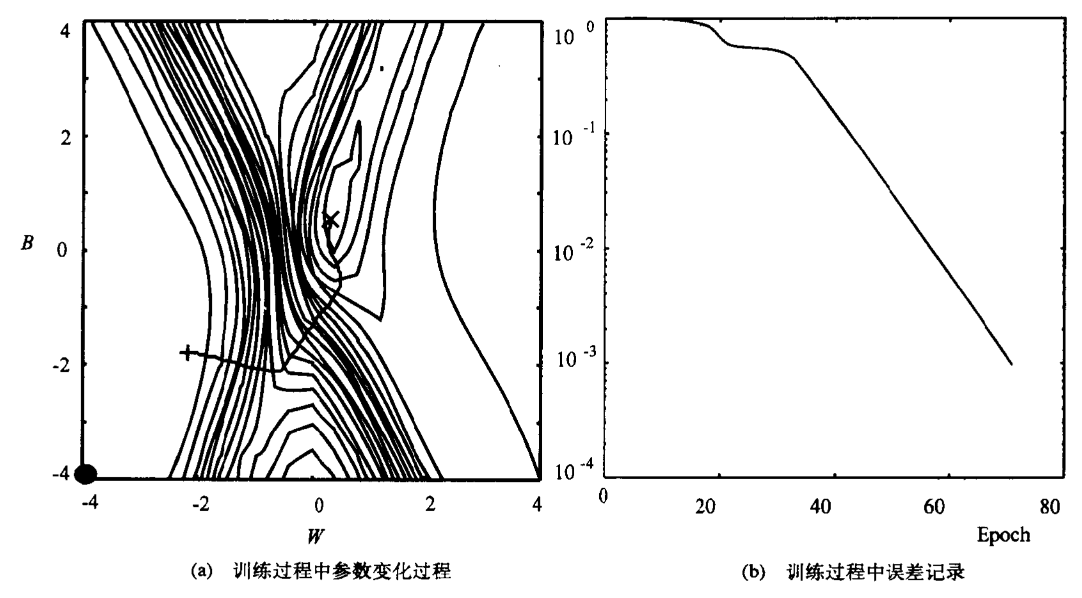


图 4.14 误差曲面图形

图 4.15 误差的等高线图



(a) 训练过程中参数变化过程　　　(b) 训练过程中误差记录

图 4.16　训练过程中参数变化过程和误差记录

两图以不同的方式画出了误差的分布。从图 4.14 中可以看出一个较高的平原以及一个较低的扁平层出现在两个边缘。在等高线图中，平原面覆盖了左半边的大部分，而一个较大的“丘”地是在中心的底部，一个较小的“丘”在顶部。最小误差是在平面和丘之间：两角之间的交点给出了最小误差的网络权值。

训练一个网络的工作始于随机初始化的权值。本例中在误差等高线中用符号“+”画出了随机选取的误差位置。图 4.16(a)给出训练过程中误差从初始值移动到最终结果的过程记录。这个记录是由变量 errors 在整个训练过程中保存。这些误差可以用下面的命令画出：**ploterr (errors)**，如图 4.16(b)所示。注意权值变化在误差等高线图中的走向。它是与等高线呈垂直的角度穿越等高线向最小值逼近的。

4.4.2　隐含层的神经元数

网络训练精度的提高，可以通过采用一个隐含层，而增加其神经元数的方法来获得。这在结构实现上，要比增加更多的隐含层要简单得多。那么究竟选取多少个隐含层节点才合适？这在理论上并没有一个明确的规定。在具体设计时，比较实际的做法是通过对不同神经元数进行训练对比，然后适当地加上一点余量。

为了对隐含层神经元数在网络设计时所起的作用有一个比较深入的理解，下面先给出一个有代表性的实例，然后从中得出几点结论。

【例 4.5】用两层 BP 网络实现“异或”功能。

这是一个很能说明问题的例子。我们已经知道，单层感知器不可能实现“异或”功能，即使网络实现如下的输入/输出的功能：

$$P=\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 1 & 0 & 1\end{bmatrix},\quad T=[0 \quad 1 \quad 1 \quad 0]$$

很明显，希望在输入平面上，用一条直线将目标矢量的四个点分成两部分是不可能的，但两条直线则可以解决的问题。

当然，三条直线、四条直线均能解决此问题。另外我们知道，对于一个二元输入网络来说，神经元数即为分割线数。所以，采用 2,3 和 4 等数作为隐含层神经元均能解决此问题。由此，可以画出实现“异或”功能的四种可能的 BP 网络结构图。

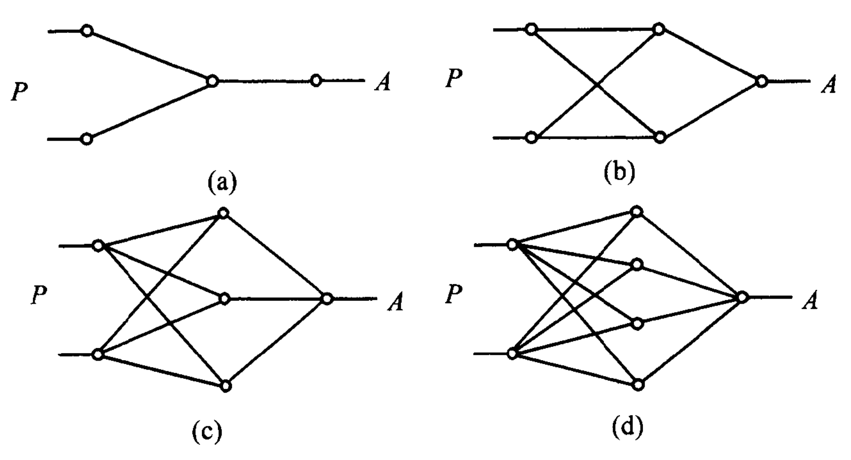


图 4.17 s1 分别为 1, 2, 3 和 4 时的求解“异或”功能的网络结构

图 4.17(a)中，输出节点与隐含层节点相同，显然该输出层是多余的，该网络也不能解决问题，因此需要增加隐含层的节点数。

更加极端的情况是多于 4 个隐含层的节点时，会有什么结果？在此例中，隐含层中神经元数为多少时为最佳？为了弄清这些问题我们针对 $s1 = 2, 3, 4, 5, 6$ 以及为 20 、 25 和 30 时对网络进行了设计。为了节省训练时间，选择误差目标为 err_goal = 0.02，并通过对网络训练时所需的循环次数和训练时间的情况来观察网络求解效果。各网络的训练结果如表 4.1 所示。

图 4.18 给出了表 4.1 所对应的训练误差的记录。从中可以看到其误差曲线下降的过程。撇开初始值的影响，误差在 $s1$ 较少时显得比较平，且下降速度也慢，而随着 $s1$ 的增大，下降速度增快。不过，由于隐含层节点数的增加，其初始误差值随之增加，这一点从图 4.18(b)中可以清楚地看出。

表 4.1 $s1 = 2, 3, 4, 5, 6, 20, 25$ 和 30 时的网络训练结果

$s1$	时间(秒)	循环次数
2	5.71	118
3	4.40	90
4	4.39	88
5	4.45	85
6	4.62	85
20	3.57	68
25	4.06	72
30	5.11	96

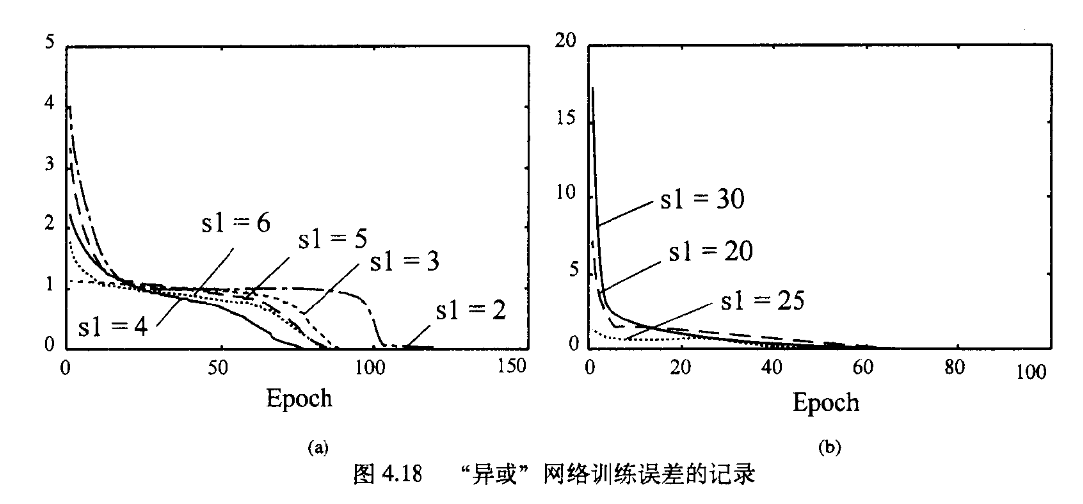


图 4.18 “异或”网络训练误差的记录

我们评价一个网络设计的好坏，首先是它的精度，再一个就是训练时间。时间包含有两层：一是循环次数，二是每一次循环中计算所花费的时间。

从表 4.1 和图 4.18 中可以看出下面几种情况：

1 ）神经元太少，网络不能很好地学习，需要训练的次数也多，训练精度也不高，如取 $s1 = 2$ 时，虽然能够解决问题，但为最糟糕的情况，因为很难达到较高的训练精度；

2 ）一般而言，网络隐含层神经元的个数 $s1$ 越多，功能越大，但当神经元数太多，一是循环次数，也就是训练时间随之增加。另外，还会产生其他问题。实际上，在进行函数逼近时，$s1$ 过大，可能导致不协调的拟合。表 4.1 中当取 $s1$ 大于 25 以后，网络解决问题的能力就开始出现问题；

3 ）当 $s1 = 3, 4, 5$ 时，其输出精度都相仿，而 $s1 = 3$ 时的训练次数最多。$s1$ 的增加能够加速误差的下降，不过从另一方面来看，随着 $s1$ 的增加，在每一次循环过程中所要进行的计算量也随着增加，所以所需要的训练时间并不一定也随之减少，这一点从 $s1 = 5$ 和 $s1 = 6$ 的情况中看得很清楚：两者所用训练时间相同，而后者比前者所用训练时间较长。对于本题，可以说选 $s1 = 3, 4, 5$ ，直至 15 均可，当然以 $s1 = 5$ 和 6 时为最佳。由此可以看出，网络隐含层节点数的选择是有一个较广的范围的。不过从网络实现的角度上说，倾向于选择较少的节点数。一般地讲，网络 $s1$ 的选择原则是：在能够解决问题的前提下，再加上 1 到 2 个神经元以加快误差的下降速度即可。

4.4.3 初始权值的选取

由于系统是非线性的，初始值对于学习是否达到局部最小、是否能够收敛以及训练时间的长短的关系很大。如果初始权值太大，使得加权后的输入和 n 落在了 S 型激活函数的饱和区，从而导致其导数 $f'(s)$ 非常小，而在计算权值修正公式中，因为 $\delta \propto f'(n)$，当 $f'(n) \to 0$ 时，则有 $\delta \to 0$。这使得 $\Delta w_{ij} \to 0$，从而使得调节过程几乎停顿下来。所以，一般总是希望经过初始加权后的每个神经元的输出值都接近于零，这样可以保证每个神经元的权值都能够在它们的 S 型激活函数变化最大之处进行调节。所以，一般取初始权值在(-1,1)之间的随机数。另外，为了防止上述现象的发生，威得罗等人在分析了两层网络是如何对一

个函数进行训练后，提出一种选定初始权值的策略：选择权值的量级为$\sqrt[s1]{S1}$，其中$s1$为第一层神经元数目。利用他们的方法可以在较少的训练次数下得到满意的训练结果。在MATLAB工具箱中可采用函数**nwlog.m**或**nwtan.m**来初始化隐含层权值W1和B1。其方法仅需要使用在第一隐含层的初始值的选取上，后面层的初始值仍然采用随机取数。

【例4.6】较好的初始值时的训练效果的观察。

以前面的【例4.1】为例，当改用下列初始值：

```
[ W1,B1 ] = nwtan(S1,R);
[ W2,B2 ] = rands(S2,S1)*0.5;
```

在这个初始值函数下，获得的一组初始值为：

```
W1 = [3.5 3.5 3.5 3.5 3.5 3.5 ];
B1 = [ -2.8562; 1.0774; -0.5880; 1.4083; 2.8722 ];
W2 = [0.2622 -0.2375 -0.4525 0.2361 -0.1718 ];
B2 = [0.1326 ];
```

重新训练网络后，相对于原先随机初始值时的6 801次的训练，仅用了454次，就达到了同样的目标误差0.02 (0.0198409)。这比标准的反向传播法快了10倍以上，明显地节省了大量的训练时间。图4.19和图4.20给出了初始权值时的输入/输出图以及网络训练结束后的误差记录。网络训练结束后的权值为：

```
W1 = [ 3.6149 3.8072 3.6115 3.3737 3.7562];
B1 = [ -2.7713; 1.1006; -0.7192; 1.5936; 2.9815];
W2 = [-0.5747 -1.1234 0.8447 -0.0467 1.3260];
B2 = -0.9116
```

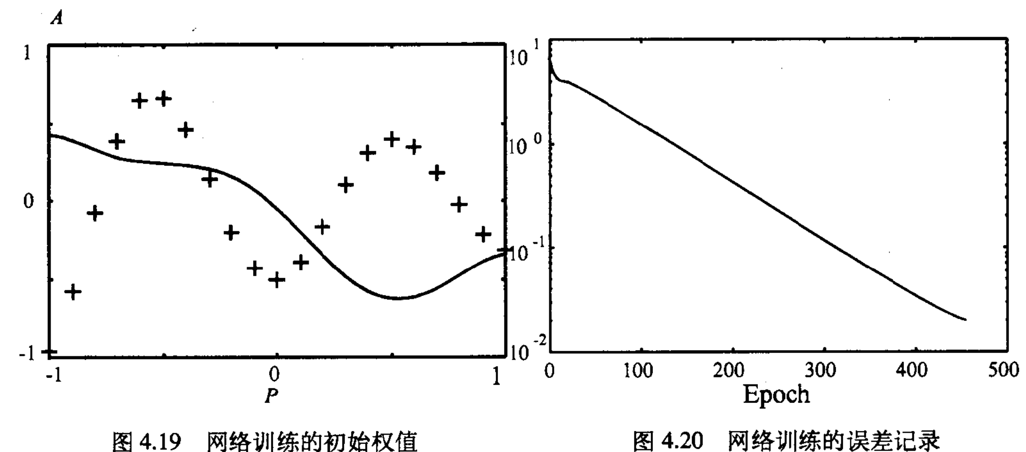


图4.19 网络训练的初始权值　　　　图4.20 网络训练的误差记录

4.4.4 学习速率

学习速率决定每一次循环训练中所产生的权值变化量。大的学习速率可能导致系统的不稳定；但小的学习速率导致较长的训练时间，可能收敛很慢，不过能保证网络的误差值不跳出误差表面的低谷而最终趋于最小误差值。所以在一般情况下，倾向于选取较小的学

习速率以保证系统的稳定性。学习速率的选取范围在 0.01 ~ 0.8 之间。

和初始权值的选取过程一样，在一个神经网络的设计过程中。网络要经过几个不同的学习速率的训练，通过观察每一次训练后的误差平方和 Σe^2 的下降速率来判断所选定的学习速率是否合适。如果 Σe^2 下降很快，则说明学习速率合适，若 Σe^2 出现振荡现象，则说明学习速率过大。对于每一个具体网络都存在一个合适的学习速率。但对于较复杂网络，在误差曲面的不同部位可能需要不同的学习速率。为了减少寻找学习速率的训练次数以及训练时间，比较合适的方法是采用变化的自适应学习速率，使网络的训练在不同的阶段自动设置不同学习速率的大小。这一方法将在后面讨论。

【例 4.7】观察学习速率太大的影响。

具有非线性激活函数的各层网络对大的学习速率很敏感。对于非线性网络，不像线性网络那样可以选择一个最优学习速率。层数越多，网络训练的学习速率只能越低。较大的学习速率极易产生振荡而使其很难达到期望的目标，有的甚至发散。为了演示使用太大学习速率的后果，我们仍然采用简单的【例 4.2】的输入/目标输出来进行观察和对比。取学习速率为:

lr = 4;

再次训练网络并观察其训练的不同权值在误差等高线图中的轨迹，如图 4.21 所示。图 4.22 给出学习速率的记录，图中纵轴表示学习速率，横轴为训练次数。从中可以看到，较大的学习速率在训练初始阶段并不成问题，且能够加速误差的减少，能比一般的训练学习速率产生更佳的误差减小率。但是随着训练的不断深入则出现了问题。由于学习速率过大，使网络每一次的修正值太大，从而导致在权值的修正过程中超出误差的最小值而永不收敛。

避免这一情况发生的办法就是减少学习速率。当然，经验在选取学习速率时是很有帮助的。再就是采用后面将要讲到的自适应学习速率，使网络根据不同的训练阶段自动调节其学习速率。

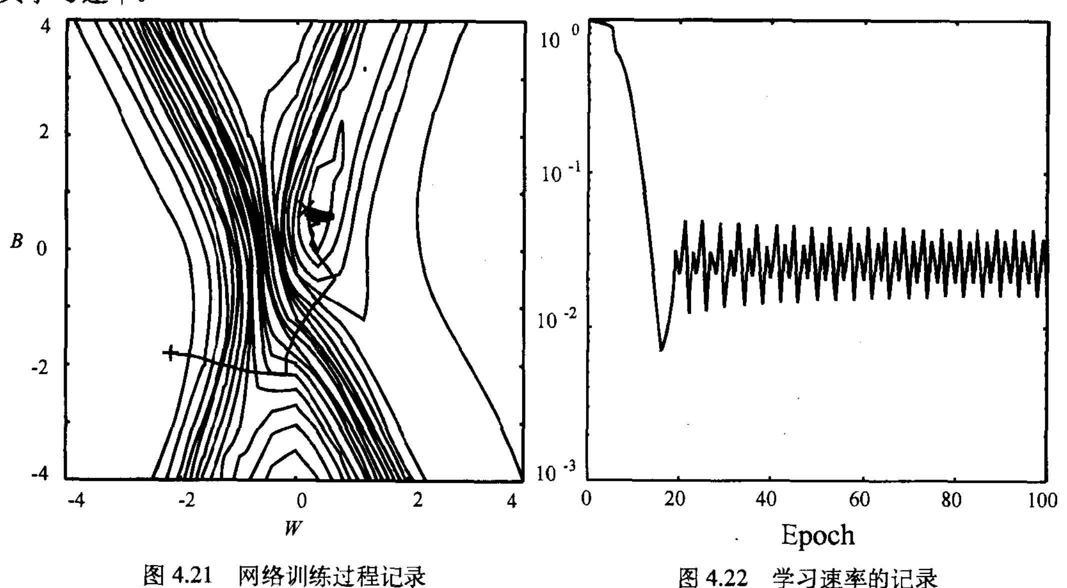


图 4.21　网络训练过程记录　　图 4.22　学习速率的记录

4.4.5 期望误差的选取

在设计网络的训练过程中，期望误差值也应当通过对比训练后确定一个合适的值，这个所谓的"合适"，是相对于所需要的隐含层的节点数来确定，因为较小的期望误差值是要靠增加隐含层的节点，以及训练时间来获得的。一般情况下，作为对比，可以同时对两个不同期望误差值的网络进行训练，最后通过综合因素的考虑来确定采用其中一个网络。

4.5 限制与不足

虽然反向传播法得到广泛的应用，但它也存在自身的限制与不足，其主要表现在于它的训练过程的不确定上。具体说明如下：

（1）需要较长的训练时间

对于一些复杂的问题，BP 算法可能要进行几小时甚至更长的时间的训练。这主要是由于学习速率太小所造成的。可采用变化的学习速率或自适应的学习速率来加以改进。

（2）完全不能训练

这主要表现在网络出现的麻痹现象上。在网络的训练过程中，当其权值调得过大，可能使得所有的或大部分神经元的加权总和 n 偏大，这使得激活函数的输入工作在 S 型转移函数的饱和区，从而导致其导数 $f'(n)$ 非常小，从而使得对网络权值的调节过程几乎停顿下来．通常为了避免这种现象的发生，一是选取较小的初始权值，二是采用较小的学习速率，但这又增加了训练时间。

（3）局部极小值

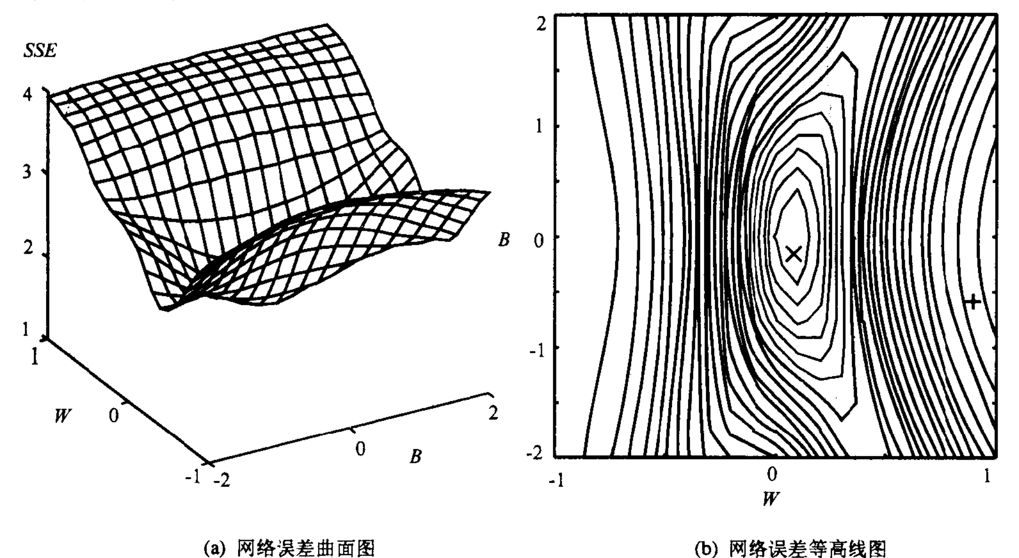
BP 算法可以使网络权值收敛到一个解，但它并不能保证所求为误差超平面的全局最小解，很可能是一个局部极小解。这是因为 BP 算法采用的是梯度下降法，训练是从某一起始点沿误差函数的斜面逐渐达到误差的最小值。对于复杂的网络，其误差函数为多维空间的曲面，就像一个碗，其碗底是最小值点。但是这个碗的表面是凹凸不平的，因而在对其训练过程中，可能陷入某一小谷区，而这一小谷区产生的是一个局部极小值。由此点向各方向变化均使误差增加，以致于使训练无法逃出这一局部极小值。

如果对训练结果不满意的话，通常可采用多层网络和较多的神经元，有可能得到更好的结果。然而，增加神经元和层数，同时增加了网络的复杂性以及训练的时间。在一定的情况下可能是不明智的。可代替的办法是选用几组不同的初始条件对网络进行训练，以从中挑选它们的最好结果。

【例 4.8】误差的局部和全局最小值的观察。

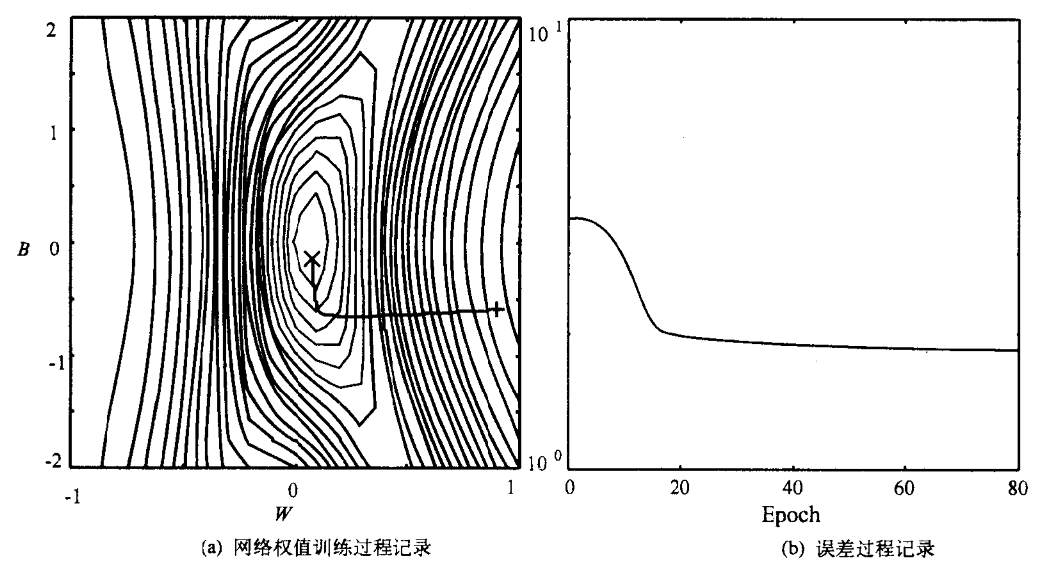
非线性网络引进了线性网络所没有的复杂性。线性网络在其误差曲面上只有唯一的一个最小值，而非线性网络可能存在几个局部极小值。这些极小值是各不相同的。好比在同一个海拔上有几个深度不同的峡谷。理想的网络应当具有最小误差的网络，即全局误差最小的网络。但所采用的基于梯度下降法的 BP 算法并不能保证做到这一点。网络有可能被陷入到误差曲面的一个较高的谷中，即局部极小值点。

这里沿用【例 4.3】中的输入矢量和目标矢量来观察误差的局部与全局最小值的问题。其误差曲面以及误差的等高线图形如图 4.23 所示。从图中可看到这两个误差表面在图的中心有全局最小值，谷的两边均有局部极小值。这个情况在误差等高线上的右顶部和右底部能够清楚地找到。初始权值为: W0 = 0.8951; B0 = -0.5897 。等高线图形中的 "+"表示初始权值时的误差值。用其开始训练网络如图 4.24(a)轨迹中可以看到，网络沿着误差梯度直落到全局最小值。若全局最小值比期望误差值小，那么即可解决问题。图 4.24(b)给出了训练过程中的误差记录。



(a) 网络误差曲面图　　(b) 网络误差等高线图

图 4.23　网络误差曲面图与等高线图

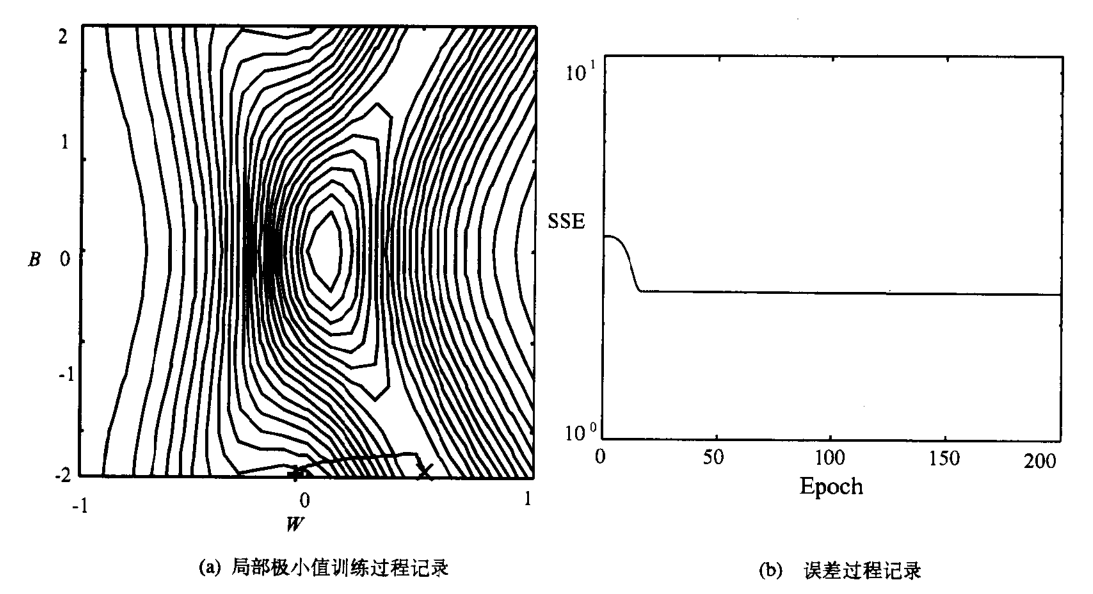


(a) 网络权值训练过程记录　　(b) 误差过程记录

图 4.24　网络权值训练过程及其误差记录

当网络选取不同的初始值，如: W0 = -0.0511;　　B0 = -2.4462 ; 则导致不同的训练结果如图 4.25 所示。

此时，网络达不到全局最小值，而是落入了谷中的一个局部极小值。由于这个谷是很浅的，所以导致网络的学习减速，最终达到局部极小值。



(a) 局部极小值训练过程记录　　(b) 误差过程记录

图 4.25　网络权值变化过程及其误差记录

我们如何来对待并解决局部极小值的问题？希望产生一个完全避免极小值的算法至今还没有可能。一个较现实的做法是，训练网络使其具有足够低的可接受的误差。这样，无论网络是否落入局部极小值，问题都得到了解决。可行方法之一就是采用比所需要的更强大的网络来解决问题。例如，增加较多的神经元层，较多的神经元数，这些都常常有助于达到期望的误差。另一个可用来跳出较浅的凹凸不平区域的局部极小值的技术是下面要介绍的附加动量法。

4.6　反向传播法的改进方法

由于在人工神经网络中，反向传播法占据了非常重要的地位，所以近十几年来，许多研究人员对其做了深入的研究，提出了很多改进的方法。主要目标是为了加快训练速度，避免陷入局部极小值和改善其它能力。本节只讨论前两种性能的改进方法的有关内容。

4.6.1　附加动量法

附加动量法使网络在修正其权值时，不仅考虑误差在梯度上的作用，而且考虑在误差曲面上变化趋势的影响，其作用如同一个低通滤波器，它允许网络忽略网络上的微小变化特性。在没有附加动量的作用下，网络可能陷入浅的局部极小值，利用附加动量的作用则有可能滑过这些极小值。

该方法是在反向传播法的基础上在每一个权值的变化上加上一项正比于前次权值变化

量的值，并根据反向传播法来产生新的权值变化。带有附加动量因子的权值调节公式为:

$$\begin{aligned} \Delta w_{ij}(k+1) &= (1-mc)\eta\delta_i p_j + mc\Delta w_{ij}(k) \\ \Delta b_i(k+1) &= (1-mc)\eta\delta_i + mc\Delta b_i(k) \end{aligned} \tag{4.11}$$

其中 k 为训练次数， mc 为动量因子，一般取 0.95 左右。

附加动量法的实质是将最后一次权值变化的影响，通过一个动量因子来传递。当动量因子取值为零时，权值的变化仅是根据梯度下降法产生；当动量因子取值为 1 时，新的权值变化则是设置为最后一次权值的变化，而依梯度法产生的变化部分则被忽略掉了。以此方式，当增加了动量项后，促使权值的调节向着误差曲面底部的平均方向变化，当网络权值进入误差曲面底部的平坦区时， δ_i 将变得很小，于是， $\Delta w_{ij}(k+1) \approx \Delta w_{ij}(k)$ ，从而防止了 $\Delta w_{ij} = 0$ 的出现，有助于使网络从误差曲面的局部极小值中跳出。

在 MATLAB 工具箱中，带有动量因子的权值修正法是用函数 **learnbpm.m** 来实现的。在使用此函数之前，先需将初始权值的变化置零:

```
dW = 0*W;    dB =0*B;
```

然后，权值的变化可以根据当前层的输入（比如 P ），误差变化（ D = **deltalog.m, deltatan.m, deltalin.m** ），学习速率 lr ，以及动量因子 mc 求得:

```
[ dW, dB ] = learnbpm ( P, D, lr, mc, dW, dB );
```

函数 **learnbpm.m** 返回一个新的权值变化和偏差变化矢量。当要训练一个没有偏差或具有固定偏差的网络时， dB 项可以从函数中消失，这样，网络的偏差则不被修正。

根据附加动量法的设计原则，当修正的权值在误差中导致太大的增长结果时，新的权值应被取消而不被采用，并使动量作用停止下来，以使网络不进入较大误差曲面；当新的误差变化率对其旧值超过一个事先设定的最大误差变化率时，也得取消所计算的权值变化。其最大误差变化率可以是任何大于或等于 1 的值。典型的值取 1.04 。所以在进行附加动量法的训练程序设计时，必须加进条件判断以正确使用其权值修正公式。

训练程序中对采用动量法的判断条件为:

$$mc = \begin{cases} 0 & \text{当 } SSE(k) > SSE(k-1)\cdot 1.04 \\ 0.95 & \text{当 } SSE(k) < SSE(k-1) \\ mc & \text{其他} \end{cases} \tag{4.12}$$

所有这些判断过程细节均包含在 MATLAB 工具箱中的函数 **trainbpm.m** 中，它可以训练一层直至三层的带有附加动量因子的反向传播网络。以调用其他函数的同样方式调用 **trainbpm.m** ，只是对变量 TP 需提供较多的参数。下面是对单层网络使用函数 **trainbpm.m** 的情形:

```
[ W, B, epochs, errors ] = trainbpm ( W, B, 'F', P, T, TP );
```

TP 行矢量中的训练函数是训练过程中所需用到的参数，它们依次为：显示结果的频率 disp_freq ，期望的误差目标 err_goal ，学习速率 lr ，动量因子 mc 以及最大误差变化率

err_ratio。

为了能够观察附加动量法的作用效果，我们特地选取了一层网络的训练作为例子，并且令其偏差固定不变。在训练过程中，通过绘出权值与输出误差的函数变化图形显示出每训练一次后网络输出误差的走向，以动态变化的形式形象地让读者感受到附加动量在网络训练过程中所产生的作用。另外，在训练过程中，由此可以观察到任意初始值下的训练过程。在实际网络的设计过程中，除非网络比较简单，加上经验比较丰富，一般情况下是不直接采用网络的训练函数来训练网络，因为训练函数只有在整个训练完成之后才给出结果。对于复杂网络，在其训练过程中，往往可能会由于一些参数选取不当，如学习速率可能太大等原因，而使得训练不合适，如果设置的训练次数较大，如几千次，在这种情况下，花了很长时间获得的训练结果很可能是无效的。较好的做法是写出训练过程，并在其中加上监视作图程序，这样可以使设计者在发现由于参数设计不当而进行无用训练时及时中止训练而进行调整与改进。

【例 4.9】采用附加动量法的反向传播网络的训练。

下面以【例 4.3】数据为例编写出带有附加动量的反向传播法以及具有上述观察效果的 MATLAB 程序。

```
% bp9.m
%
clf reset
pausetime = 0.1
% 初始化及赋初值
P = [ -6.0 -6.1 -4.1    -4.0    5.0    -5.1 6.0 6.1 ];
T = [    0    0    0.97 0.99    0.01 0.03 1.0 1.0 ];
[ R, Q ] =size(P);    [ S, Q ] = size(T);
disp('The bias B is fixed at 3.0 and will not learn.')
Z1 = meun('Intialize Weight with:', ...
    'W0 = [ -0.9 ]; B0 = 3;', ...                   % 按给定初始值
    'Pick Values with Mouse/Arrow Keys', ...        % 用鼠标在图上任点初始值
    'Random Intial Condition [Default];             % 随机初始值（缺省情况）
disp('')
B0 = 3;
if Z1 = 1    W0 = [ -0.9 ];
elseif Z1 = =3    W0 = rand (S,R);
end
Z2 menu ('Use momentium constant of :', ...
    '0.0', ...
    '0.95 [ Default ]');
disp('')
if Z2 == 2    [ W0, dummy ] = ginput(1);
```

```
end
% 作权值一误差关系图并标注初始值
A = logsig(W0*P,B0);   E = T - A;      SSE = sumsqr(E):
h = plot (W0,SSE,'.m');
set (h,'markersize',12);
pause2(pausetime);
hold on
% 训练网络
disp_fqre = 5; max_epoch = 500; err_goal = 0.01;     lr = 0.05;
if Z2 ==1, momentum = 0; else momentum = 0.98; end
err_ratio = 1.04;    W = W0; B = B0;
A = logsig(W0*P,B0);   E = T - A;      SSE = sumsqr(E):
mc = 0; dW = 0*dW;                                            % 初始化动量因子
for epoch = 1 : max_epoch
if SSE < err_goal, epoch = epoch - 1; break, end
D = deltalog ( A, E ); dW = learnbpm (P,D,lr,mc,dW); TW = W + dW;
TA = logsig (TW*P,B); TE = T - TA; TSSE = sumsqr(TE);
if TSSE > SSE*err_ratio, mc = 0;                              % 判断动量效果
else if TSSE < SSE, mc = momentum; end
W = TW; A = TA; E = TE; SSE = TSSE; end
errors = [ error SSE ];
% 显示结果
plot(W,SSE,'xg');pause,hold off
ploterr(errors),pause
W,B
SSE = sumsqr(T-logsig(W*P,B))
end
```

一维误差曲线图如图 4.26 所示。可以看到在误差曲线上有两个误差最小值，一个为局部极小值在左边，右边的为全局最小值。本例中的缺省初始值为 W0 = -0.9，并表示在图的左边以 " o"表示。如果动量因子 mc 取为 0，网络则以纯梯度法进行训练，此举的结果如图 4.27 所示。其误差的变化趋势是以简单的方式“滚到”局部极小值的底部就再也停止不动了。图 4.28 给出了误差记录。

当采用附加动量法后，网络的训练则可以自动地避免陷入这个局部极小值。这个结果如图 4.29 所示。

网络的训练误差先落入局部极小值，在附加动量的作用下，继续向前产生一个正向斜率的运动，并跳出较浅的峰值，落入了全局最小值。然后，仍然在附加动量的作用下，达到一定的高度后（即产生了一个 SSE > SSE*1.04 ）自动返回，并像弹子滚动一样来回左右摆动，直至停留在最小值点上。

图 4.26　网络误差曲线图

图 4.27　mc = 0 时的训练结果

图 4.28　训练误差记录

图 4.29　采用附加动量法的训练结果

读者可以执行上面的程序来观察带有附加动量法的网络训练的全过程，并可以任意选择初始权值来观察其训练结果。

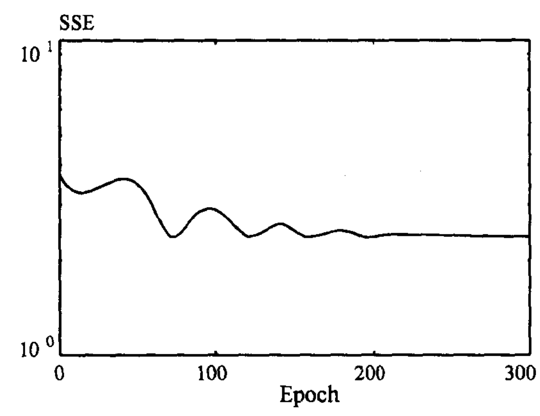


图 4.30　采用动量法时的训练误差记录



通过观察可以发现，训练参数的选择对训练效果的影响是相当大的。如学习速率太大，将导致其误差值来回振荡；学习速率太小，则导致太小动量能量，从而使其只能跳出很浅的“坑”。对于较大的“坑”或“谷”将无能为力。而从另一方面来看，其误差曲线（面）

的形状与凹凸性是由问题的本身决定的，所以每个问题都是不相同的。这必然对学习速率的选择带来了困难。一般情况下只能采用不同的学习速率进行对比尝试（典型的值取为 0.05 ）。

通过观察带有附加动量法的网络训练的过程还使我们感到，对于这种训练必须给予足够的训练次数，以使其训练结果为最后稳定到最小值时的结果，而不是得到一个正好摆动到较大误差值时的网络权值。

此训练方法也存在有缺点。它对训练的初始值有要求，必须使其值在误差曲线上的位置所处误差下降方向与误差最小值的运动方向一致。如果初始误差点的斜率下降方向与通向最小值的方向背道而驰，则附加动量法失效。训练结果将同样落人局部极小值而不能自拔。初始值选得太靠近局部极小值时也不行，所以建议多用几个初始值先粗略训练几次以找到合适的初始位置。另外，学习速率太小也不行，比如对于本例题，选择 lr = 0.01 ，网络则没有足够的能量跳过低“谷”。

4.6.2 误差函数的改进

前面定义的函数是一个二次函数:

$$E=\frac{1}{2}\ _{k}\ (t_k - a_k)^2$$

当 a_k 趋向 1 时， E 趋向一个常数，即处于 E 的平坦区，从而造成了不能完全训练的麻痹现象。所以当一个网络的误差曲面存在着平坦区时，可以选用别的误差函数 $f(t_k,a_k)$ 来代替 $(t_k - a_k)^2$ 的形式，只要其函数在 $a_k = t_k$ 时能够达到最小值即可。

包穆(Baum)等人于 1988 年提出一种误差函数为:

$$E=\sum_k \left[\frac{1}{2}(1+t_k)\log\frac{1+t_k}{1+a_k}+\frac{1}{2}(1-t_k)\log\frac{1-t_k}{1-a_k}\right]$$

该式同样满足当 $a_k = t_k$ 时， $E=0$ ，不过，当 $a_k \to \pm 1$ 时，该式发散。所以能够克服麻痹现象。如果采用双曲正切函数来作为激活函数，即取:

$$f(n)=\tanh(n)=\frac{1-e^{-2n}}{1+e^{-2n}}=\frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$$

又因为

$$f^2(n)=\tanh^2(n)=\frac{e^{2n}-2e^{n}e^{-n}+e^{-2n}}{(e^{n}+e^{-n})^2}=\frac{-2}{(e^{n}+e^{-n})^2}$$

而

$$f\ (n)=\tanh(n)=\frac{(e^{n}+e^{-n})(e^{n}+e^{-n})-(e^{n}-e^{-n})(e^{n}-e^{-n})}{(e^{n}+e^{-n})^2}$$

$$=\frac{4}{(e^{n}+e^{-n})^2}=\frac{(e^{n}+e^{-n})^2-2}{(e^{n}+e^{-n})^2}=1-f^2(n)$$

求误差函数对输出层的变量 n 求一阶导数并同时考虑关系式：$a_k' = 1 - a_k^2$：

$$\frac{\partial E}{\partial n} = \frac{1}{2}(1+t_k)\frac{1+a_k}{1+t_k}\frac{(1+t_k)(-a_k)}{(1+a_k)^2}(1-a^2) + \frac{1}{2}(1-t_k)\frac{1-a_k}{1-t_k}\frac{(1-t_k)a_k}{(1-a_k)^2}(1-a^2)$$

$$= t_k - a_k = \delta_{ki}$$

与常规的误差函数的情况 $\delta_{ij} = f'(n)(t_k - a_k)$ 相比较，其中的 $f'(n)$项消失了。这样，当 n 增大，进入激活函数的平坦区，使 $f'(n) \to 0$ 时，不会产生不能完全训练的麻痹现象。但由于失去了 $f'(n)$对 Δw 的控制作用，过大的 Δw 又有可能导致网络过调或振荡。为了解决这个问题，1989 年，范尔曼(S. Fahlman)提出一种折中的方案，即取 $\delta_k = [f'(n)+0.1](t_k - a_k)$，该式一方面恢复了 $f'(n)$的某些影响，另一方面当 $|n|$ 变大时，仍能保持 δ_k 有一定的大小，从而避免了麻痹现象的发生。

4.6.3 自适应学习速率

对于一个特定的问题，要选择适当的学习速率不是一件容易的事情。通常是凭经验或实验获取，但即使这样，对训练开始初期功效较好的学习速率，不见得对后来的训练合适。为了解决这一问题，人们自然会想到在训练过程中，自动调整学习速率。通常调节学习速率的准则是：检查权值的修正值是否真正降低了误差函数，如果确实如此，则说明所选取的学习速率值小了，可以对其增加一个量；若不是这样，而产生了过调，那么就应该减小学习速率的值。下式给出了一种自适应学习速率的调整公式：

$$\eta(k+1) = \begin{cases} 1.05\eta(k) & SSE(k+1) < SSE(k) \\ 0.7\eta(k) & SSE(k+1) > 1.04\ SSE(k) \\ \eta(k) & \text{其他} \end{cases} \tag{4.13}$$

初始学习速率 $\eta(0)$ 的选取范围可以有很大的随意性。

【例 4.10】采用自适应学习速率训练网络。

用【例 4.6】中所提出的初始条件法能够产生比【例 4.1】快得多的网络训练结果，说明采用较好的权值和偏差的初始值，能够加快训练速率。如果再加用自适应学习速率，则能够更进一步地减少训练时间。

与采用附加动量法时的判断条件相仿，当新误差超过旧误差一定的倍数时，学习速率将减少；否则其学习速率保持不变；当新误差小于旧误差时，学习速率将被增加。此方法可以保证网络总是以最大的可接受的学习速率进行训练。当一个较大的学习速率仍能够使网络稳定学习，使其误差继续下降，则增加学习速率，使其以更大的学习速率进行学习。一旦学习速率调得过大，而不能保证误差继续减少，则减少学习速率直到使其学习过程稳定为止。

MATLAB 工具箱中带有自适应学习速率进行反向传播训练的函数为:**trainbpa.m** 。它可以训练直至三层网络。使用方法为:

[W,B,epochs,TE] = trainbpa(W,B,'F',P,T,TP)

在行矢量 TP 中的参数依次为：显示频率 disp_freq，最大训练次数 max_epoch，目标误差 err_goal，初始学习速率 lr，递增乘因子 lr_inc，递减乘因子 lr_dec 和误差速率 err_ratio。

函数在训练结束后返回最终权值 W 和偏差 B ，训练网络所用次数 epochs 和训练误差记录 TE 。 TE 是两个行矢量，第一行为网络的训练误差，第二行为所对应的学习速率。

同其他训练函数的调用方法一样，这个训练过程函数的应用非常简单，整个网络的设计训练过程只需要以下几行程序：

```
disp_freq = 10;
max_epoch = 2000;
err_goal =0.02;
lr = 0.02;
lr =_inc = 1.05;
lr_idec = 0.7;
err_ratio = 1.04;
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec err_ratio];
[W1,B1,W2,B2,epochs,TE] = trainbpa(W1,B1, 'tansig ',W2,B2, 'purelin',P,T,TP];
```

仅训练了 120 次就达到了目标误差 0.02 的目的。这个结果只是采用较好初始条件情况下的 **454** 次训练的近四分之一倍。它同时是固定学习速率为 0.01 时的五十分之一(【例 4.1 】中的训练次数为 6801)。

图 4.31 给出了学习速率在训练过程中的记录。误差平方和的记录在图 4.32 中。

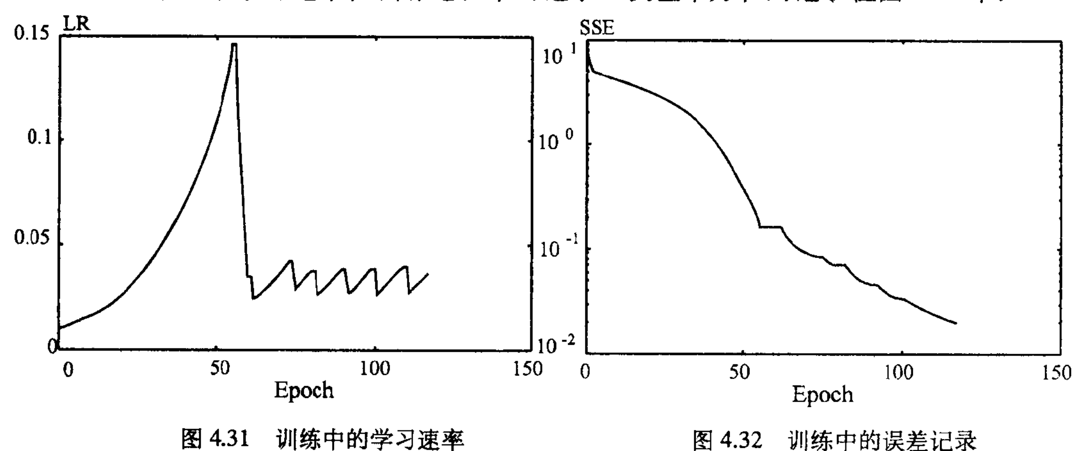


图 4.31　训练中的学习速率　　　　图 4.32　训练中的误差记录

从中可以看出：在训练的初始阶段，学习速率是以直线形式上升。当接近误差最小值时，学习速率又自动地下降。注意误差曲面有少许的波动。

可以将动量法和自适应学习速率结合起来以利用两方面的优点。这个技术已编入了函数 **trainbpx.m** 之中。这个函数的调用和其他函数一样，只是需要更多的初始参数而已：

```
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec mom_const err_ratio];
[ W,B,epochs,[errors;lr]] = trainbpx ( W,B,F,P,T,TP )
```

4.6.4 双极性 S 型压缩函数法

另外一种改进训练性能的简单方法为双极性S型压缩函数反向传播法。一般对数S型压缩函数的输出动态范围为(0,1)，这不是最佳的，因为从权值调节公式可知，权值的变化也正比于前一层的输出，而因其中一半是趋向0的一边，这必然引起权值调节量的减少或不调节，从而加长了训练时间。为了解决这个问题，可将输入范围变为 1/2，同时也使S型函数置偏的输出范围也变为 ± 1/2，即由

$$a_{原}=\frac{1}{1+e^{-n}}$$

改为

$$a_{新}=-\frac{1}{2}+\frac{1}{1+e^{-n}}$$

要注意的是，当利用反向传播法时，因为

$$a^2=(-\frac{1}{2}+\frac{1}{1+e^{-n}})^2=\frac{1}{4}-\frac{1}{1+e^{-n}}+(\frac{1}{1+e^{-n}})^2,$$

激活函数的一阶导数此时为:

$$a=\frac{-e^{-n}}{(1+e^{-n})^2}=\frac{-(-1+1+e^{-n})}{(1+e^{-n})^2}=\frac{1}{(1+e^{-n})^2}-\frac{1}{1+e^{-n}}$$

$$=\frac{1}{4}-[\frac{1}{4}-\frac{1}{1+e^{-n}}+(\frac{1}{1+e^{-n}})^2]=\frac{1}{4}-a^2$$

实验证明，采用此方法，收敛时间平均可以减少 30~50%。当然，若采用此方法来训练网络，其训练程序需要设计者自己编写一部分。

4.7 本章小结

1 ）反向传播法可以用来训练具有可微激活函数的多层前向网络以进行函数逼近，模式分类等工作;

2 ）反向传播网络的结构不完全受所要解决的问题所限制。网络的输入神经元数目及输出层神经元的数目是由问题的要求所决定的，而输入和输出层之间的隐含层数以及每层的神经元数是由设计者来决定的;

3 ）已证明，两层 S 型/线性网络，如果 S 型层有足够的神经元，则能够训练出任意输入和输出之间的有理函数关系;

4 ）反向传播法沿着误差表面的梯度下降，使网络误差最小，网络有可能陷入局部极小值;

5 ）附加动量法使反向传播减少了网络在误差表面陷入低谷的可能性并有助于减少训练时间;

6 ）太大的学习速率导致学习的不稳定，太小值又导致极长的训练时间。自适应学习速率通过在保证稳定训练的前提下，达到了合理的高速率，可以减少训练时间;

7 ）80%~90% 的实际应用都是采用反向传播网络的。改进技术可以用来使反向传播法更加容易实现并需要更少的训练时间。

习　题

〖4.1〗运用将附加动量法和自适应学习速率相结合的技术的算法函数 **trainbpx.m** 训练【例 4.1】并与其他方法的训练结果相比较。

〖4.2〗根据所学过的 BP 网络设计及改进方案设计实现模糊控制规则为 $T = 1/2*（e + ec）$ 的模糊神经网络控制器。设计要求为:

①输入、输出矢量及问题的阐述;

②给出网络结构;

③学习方法（包括所采用的改进方法）;

④初始化及必要的参数选取;

⑤最后的结果，循环次数，训练时间，其中着重讨论:

a)不同隐含层 S1 时的收敛速度与误差精度的对比分析，

b)当 S1 设置为较好的情况下，在训练过程中取始终不变的学习速率 lr 值时，对 lr 值为不同值时的训练时间，包括稳定性进行观察比较，

c)当采用自适应值学习速率时，与单一固定的学习速率 lr 中最好的情况进行对比训练的观察，

d)给出结论或体会。

⑥验证，采用插值法选取多于训练时的输入，对所设计的网络进行验证，给出验证的 A 与 T 值。

第五章　反馈网络

反馈网络（ Recurrent　Network ），又称自联想记忆网络，其目的是为了设计一个网络，储存一组平衡点，使得当给网络一组初始值时，网络通过自行运行而最终收敛到这个设计的平衡点上。

1982 年，美国加州工学院物理学家霍普菲尔德（ J.Hopfield ）发表了一篇对人工神经网络研究颇有影响的论文。他提出了一种具有相互联接的反馈型人工神经网络模型，并将“能量函数”的概念引入到对称霍普菲尔德网络的研究中，给出了网络的稳定性判据，并用来进行约束优化问题，如 TSP 问题的求解，实现 A/D 转换等。他利用多元霍普菲尔德网络的多吸引子及其吸引域，实现了信息的联想记忆（ associative memory ）功能。另外霍普菲尔德网络与电子模拟线路之间存在着明显的对应关系，使得该网络易于理解且便于实现。而它所执行的运算在本质上不同于布尔代数运算，对新一代电子神经计算机具有很大的吸引力。

反馈网络能够表现出非线性动力学系统的动态特性。它所具有的主要特性为以下两点：第一，网络系统具有若干个稳定状态。当网络从某一初始状态开始运动，网络系统总可以收敛到某一个稳定的平衡状态；第二，系统稳定的平衡状态可以通过设计网络的权值而被存储到网络中。

如果将反馈网络稳定的平衡状态作为一种记忆，那么当网络由任一初始状态向稳态的转化过程，实质上是一种寻找记忆的过程。网络所具有的稳定平衡点是实现联想记忆的基础。所以对反馈网络的设计和应用必须建立在对其系统所具有的动力学特性理解的基础上，这其中包括网络的稳定性，稳定的平衡状态，以及判定其稳定的能量函数等基本概念。

针对人工神经网络的特点，若按其运行过程的信息流向来分类，则可分为两大类：前向网络和反馈网络。在前面的章节里，主要介绍了前向网络，通过许多具有简单处理能力的神经元的相互组合作用使整个网络具有复杂的非线性逼近能力。在那里，着重分析的是网络的学习规则和训练过程，并研究如何提高网络整体非线性处理能力。在本章中，我们将集中讨论反馈网络，通过网络神经元状态的变迁而最终稳定于平衡状态,得到联想存储或优化计算的结果。在这里，着重关心的是网络的稳定性问题，研究的重点是怎样得到和利用稳定的反馈网络。

反馈式网络有多种，这里主要讨论由霍普菲尔德提出的反馈网络。霍普菲尔德网络是单层对称全反馈网络，根据其激活函数的选取不同，可分为离散型的霍普菲尔德网络（ Discrete Hopfield Neural Network，简称 DHNN)和连续型的霍普菲尔德网络（ Continuous Hopfield Neural Network，简称 CHNN ）。 DHNN 的激活函数为二值型的，其输入、输出为{0,1}的反馈网络，主要用于联想记忆， CHNN 的激活函数的输入与输出之间的关系为连续可微的单调上升函数，主要用于优化计算。

霍普菲尔德网络已经成功地应用于多种场合，现在仍常有新的应用的报道。具体的应用方向主要集中在以下方面：图像处理、语声处理、信号处理、数据查询、容错计算、模

式分类、模式识别等。

5.1 霍普菲尔德网络模型

反馈网络的网络结构如图 5.1 所示。

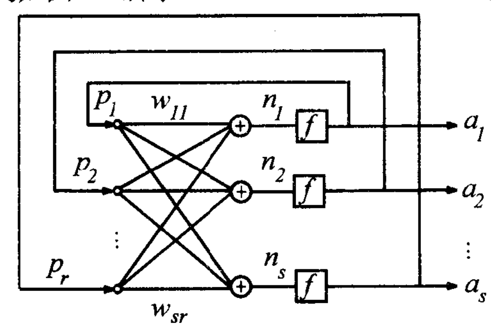


图 5.1 反馈网络结构图

该网络为单层全反馈网络，其中的每个神经元的输出都是与其他神经元的输入相连的。所以其输入数目与输出层神经元的数目是相等的，有 $r = s$ 。

在反馈网络中，如果其激活函数 $f(\cdot)$ 是一个二值型的硬函数，如图 5.2 所示，即 $a_i = \text{sgn}(n_i)$， $i = 1,2,\ldots,r$，则称此网络为离散型反馈网络，如果 $a_i = f(n_i)$ 中的 $f(\cdot)$ 为一个连续单调上升的有界函数，这类网络被称为连续型反馈网络，图 5.3 中所示为一个具有饱和线性激活函数，它满足连续单调上升的有界函数的条件，常作为连续型的激活函数。

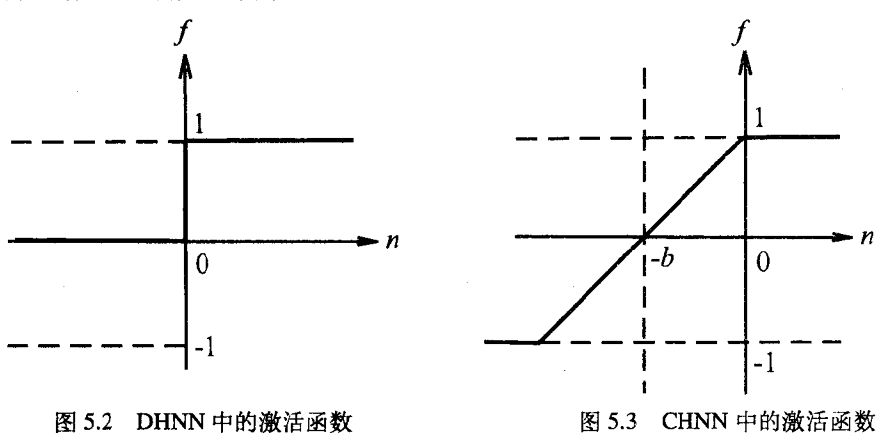


图 5.2 DHNN 中的激活函数　　图 5.3 CHNN 中的激活函数

5.2 状态轨迹

对于一个由 r 个神经元组成的反馈网络，若将加权输入和 n 视作网络的状态，则状态矢量 $N = [n_1, n_2, \ldots, n_r]^T$ ，网络的输出矢量为 $A = [a_1, a_2, \ldots, a_s]^T$ 。在某一时刻 t ，分别用

$N(t)$和 $A(t)$来表示各自的矢量。在下一时刻 $t+1$，可得到 $N(t+1)$，而 $N(t+1)$又引起 $A(t+1)$的变化，这种反馈演化的过程，使状态矢量 $N(t)$随时间发生变化。在一个 r 维状态空间上，可以用一条轨迹来描述状态变化情况。从初始值 $N(t_0)$出发，$N(t_0)+\Delta t) \rightarrow N(t_0+2\Delta t) \rightarrow \cdots \rightarrow N(t_0+m\Delta t)$，这些在空间上的点组成的确定轨迹，是演化过程中所有可能状态的集合，我们称这个状态空间为相空间。图 5.4 描述了一个三维相空间上三条不同的轨迹，对于 DHNN，因为 $N(t)$中每个值只可能为 ± 1，或{0,1}，对于确定的权值 w_{ij}，其轨迹是跳跃的阶梯式，如图中 A 所示，对于 CHNN，因为 $f(\cdot)$ 是连续的，因而，其轨迹也是连续的。如图中 B、C 所示。

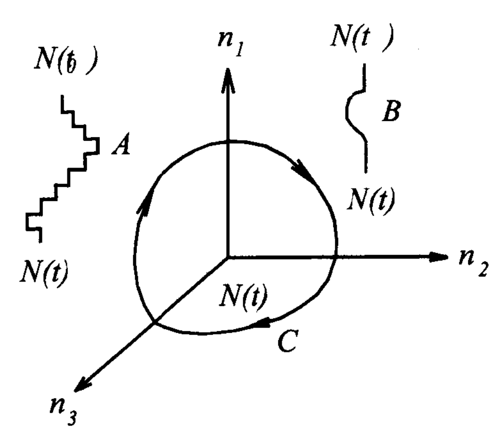


图 5.4　三维空间中的状态轨迹

对于不同的连接权值 w_{ij} 和输入 P_j $(i, j=1, 2,\cdots, r)$，反馈网络状态轨迹可能出现以下几种情况。

5.2.1　状态轨迹为稳定点

状态轨迹从系统在 t_0 时状态的初值 $N(t_0)$开始，经过一定的时间 $t\ (t>0)$后，到达 $N(t_0+t)$。如果 $N(t_0+t+\Delta t)=N(t_0+t), \Delta t>0$，则状态 $N(t_0+t)$ 称为网络的稳定点，或平衡点。由于 $N(t_0+t)$ 不再变化，对于 $P(t_0+t)$ 也达到了稳定值。即反馈网络从任一初始态 $P(0)$开始运动，若存在某一有限时刻 t，从 t 以后的网络状态不再发生变化：$P(t+\Delta t)=P(t), \Delta t>0$，则称该网络是稳定的。处于稳定时的网络状态叫做稳定状态，又称为定吸引子。

对于非线性系统来说，不同的初始值 $N(t_0)$，可能有不同的轨迹，到达不同的稳定点，这些稳定点，也可以认为是人工神经网络的解。在一个反馈网络中，存在很多稳定点，根据不同情况，这些稳定点可以分为：

1）渐近稳定点：如果在稳定点 N_e 周围的 $N(\sigma)$ 区域内，从任一个初始状态 $N(t_0)$出发的每个运动，当 $t\rightarrow\infty$ 时都收敛于 N_e，则称 N_e 为渐近稳定点。此时，不仅存在一个稳定点 N_e，而且存在一个稳定域。有时称此稳定点为吸引子，其对应的稳定域为吸引域；

2）不稳定平衡点 N_{en}：在某些特定的轨迹演化过程中，网络能够到达稳定点 N_{en}，但对于其它方向上的任意一个小的区域 $N(\sigma)$，不管 $N(\sigma)$ 取多么小，其轨迹在时间 t 以后总是偏离 N_{en}；

3）网络的解：如果网络最后稳定到设计人员期望的稳定点，且该稳定点又是渐近稳定点，那么这个点称为网络的解；

4）网络的伪稳定点：网络最终稳定到一个渐近稳定点上，但这个稳定点不是网络设计所要求的解，这个稳定点为伪稳定点。

在一个非线性的反馈网络中，存在着这些不同类型的稳定点，而网络设计的目的是希望网络最终收敛到所要求的稳定点上，并且还要有一定的稳定域。

5.2.2 状态轨迹为极限环

如果在某些参数的情况下，状态 $N(t)$的轨迹是一个圆，或一个环，状态 $N(t)$沿着环重复旋转，永不停止，此时的输出 $A(t)$也出现周期变化，即出现振荡，如图 5.4 中 C 的轨迹即是极限环出现的情形。对于 DHNN，轨迹变化可能在两种状态下来回跳动，其极限环为 2。如果在 r 种状态下循环变化，称其极限环为 r。

5.2.3 混沌现象

如果状态 $N(t)$的轨迹在某个确定的范围内运动，但既不重复，又不能停下来，状态变化为无穷多个，而轨迹也不能发散到无穷远，这种现象称为混沌(chaos)。在出现混沌的情况下，系统输出变化为无穷多个，并且随时间推移不能趋向稳定，但又不发散。这种现象越来越引起人们的重视，因为在脑电波的测试中已发现这种现象，而在真正的神经网络中存在这种现象，也应在人工神经网络中加以考虑。

5.2.4 状态轨迹发散

如果状态 $N(t)$的轨迹随时间一直延伸到无穷远，此时状态发散，系统的输出也发散。在人工神经网络中，由于输入、输出激活函数上一个有界函数，虽然状态 $N(t)$是发散的，但其输出 $A(t)$还是稳定的，而 $A(t)$的稳定反过来又限制了状态的发散。一般非线性人工神经网络中发散现象是不会发生的，除非神经元的输入/输出激活函数是线性的。

对于一个由 r 个神经元组成的反馈系统，它的行为就是由这些状态轨迹的情况来决定的。目前的人工神经网络是利用第一种情况即稳定的专门轨迹来解决某些问题的。如果把系统的稳定点视做一个记忆的话，那么从初始状态朝这个稳定点移动的过程就是寻找该记忆的过程。状态的初始值可以认为是给定的有关该记忆的部分信息，状态 $N(t)$移动的过程，是从部分信息去寻找全部信息，这就是联想记忆的过程。如果把系统的稳定点考虑为一个能量函数的极小点，在状态空间中，从初始状态 $N(t_0)=N(t_0+t)$，最后到达 N^*。若 N^*为稳定点，则可以看作是 N^*把 $N(t_0)$ 吸引了过去，在 $N(t_0)$ 时能量比较大，而吸引到 N^*时能量已为极小了。根据这个道理，可以把这个能量的极小点作为一个优化目标函数的极小点，把状态变化的过程看成是优化某一个目标函数的过程。因此反馈网络的状态移动的过程实际上是一种计算联想记忆或优化的过程。它的解并不需要真的去计算，只需要去形成一类反馈神经网络，适当地讨论其权重值 w_{ij}，使其初始输入 $A(t_0)$向稳定吸引子状态的移动就可

以达到这个目的。

霍普菲尔德网络是利用稳定吸引子来对信息进行储存的，利用从初始状态到稳定吸引子的运行过程来实现对信息的联想存取的。通过对神经元之间的权和阈值的设计，要求单层的反馈网络达到下列目标：

（1）网络系统能够达到稳定收敛

即研究系统在什么条件下不会出现振荡和混沌现象。

（2）网络的稳定点

一个非线性网络能够有很多个稳定点，对权值的设计，要求其中的某些稳定点是所要求的解。对于用做联想记忆的反馈型网络，希望稳定点就是一个记忆，那么记忆容量就与稳定点的数量有关，希望记忆的量越大，那么，稳定点的数目也越大，但稳定点数目的增加可能会引起吸引域的减小，从而使联想功能减弱。对于用做优化的反馈网络，由于目标函数（即系统中的能量函数）往往要求只有一个全局最小。那么稳定点越多，陷入局部最小的可能性就越大，因而要求系统的稳定点越少越好。

（3）吸引域的设计

希望的稳定点有尽可能大的吸引域，而非希望的稳定点的吸引域要尽可能的小。因为状态空间是一个多维空间，状态随时间的变化轨迹可能是多种形状，吸引域就很难用一个明确的解析式来表达，这在设计时要尽可能考虑。

5.3 离散型霍普菲尔德网络

5.3.1 DHNN 模型结构

在 DHNN 模型中，每个神经元节点的输出可以有两值状态，-1 或 1（0或 1），其输出类似于 MP 神经元，可表示为

$$a_i = \begin{cases} 1 & \sum_{j \neq i} w_{ij} a_i > 0 \\ -1 & \sum_{j \neq i} w_{ij} a_i < 0 \end{cases}$$

在上式中，取 $b = 0$，权矩阵中有 $w_{ij} = w_{ji}$，且取 $w_{ii} = 0$。即 DHNN 采用对称联接。因此，其网络结构可以用一个加权无向量图表示。图 5.5(a)为一个 3 节点 DHNN 结构，其中，每个输入神经元节点除了不与具有相同节点号的输出相连外，与其他节点两两相连。每个输出信号又反馈到相同的输入节点。

由图 5.5(a)，考虑到 DHNN 的权值特性 $w_{ij} = w_{ji}$，网络各节点加权输入和分别为：

$$s_1 = w_{12}a_2 + w_{13}a_3$$
$$s_2 = w_{21}a_1 + w_{23}a_3$$
$$s_3 = w_{31}a_1 + w_{13}a_2$$

由此可得简化后等效的网络结构如图 5.5(b)所示。

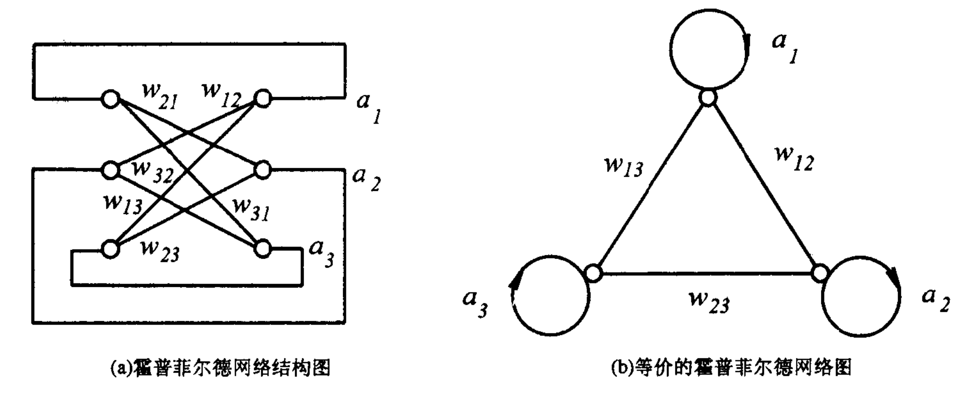


图 5.5 霍普菲尔德网络图

对于以符号函数为激活函数的网络，网络的方程可写为:

$$\left.\begin{aligned} n_j(t) &= \sum_{\substack{i=1 \\ i \neq j}}^{r} w_{ij} a_j \\ a_j(t+1) &= \mathrm{sgn}[n_j(t)] \\ &= \begin{cases} 1 & n_j(t) \geq 0 \\ -1 & n_j(t) < 0 \end{cases} \end{aligned}\right\} \tag{5.1}$$

5.3.2 联想记忆

联想记忆功能是 DHNN 的一个重要应用范围。要想实现联想记忆，反馈网络必须具有两个基本条件: ①网络能收敛到稳定的平衡状态，并以其作为样本的记忆信息; ②具有回忆能力，能够从某一残缺的信息回忆起所属的完整的记忆信息。

DHNN 实现联想记忆的过程分为两个阶段: 学习记忆阶段和联想回忆阶段。在学习记忆阶段中，设计者通过某一设计方法确定一组合适的权值，使网络记忆期望的稳定平衡点。而联想回忆阶段则是网络的工作过程。此时，当给定网络某一输人模式，网络能够通过自身的动力学状态演化过程最终达到稳定的平衡点，从而实现自联想或异联想回忆。

反馈网络有两种基本的工作方式: 串行异步和并行同步方式。

1 ）串行异步方式: 任意时刻随机地或确定性地选择网络中的一个神经元进行状态更新，而其余神经元的状态保持不变;

2 ）并行同步方式: 任意时刻网络中部分神经元（比如同一层的神经元）的状态同时更新。如果任意时刻网络中全部神经元同时进行状态更新，那么称之为全并行同步方式。

对于 s 个神经元的反馈网络 DHNN 有 2^s 个状态的可能性。其输出状态是一个包含-1 或 1 （ 0 或 1 ）的矢量，每一时刻网络将处于某一种状态下。当网络状态的更新变化采用随机异步策略，即随机地选择下一个要更新的神经元，且允许所有神经元具有相同的平均变化概率。在状态更新过程中，包括三种情况: 由-1 变为 1； 由 1 变为-1 及状态保持不变。在任一时刻，网络中只有一个神经元被选择进行状态更新或保持，所以异步状态更新的网络从某一初态开始需经过多次更新状态后才可以达到某种稳态。这种更新方式的特点是:

实现上容易，每个神经元有自己的状态更新时刻，不需要同步机制；另外，功能上的串行状态更新可以限制网络的输出状态，避免不同稳态等概率的出现；再者，异步状态更新更接近实际的生物神经系统的表现。

5.3.3 DHNN 的海布学习规则

在 DHNN 的网络训练过程中，运用的是海布调节规则：当神经元输入与输出节点的状态相同（即同时兴奋或抑制）时，从第 j 个到第 i 个神经元之间的连接强度则增强，否则则减弱。海布法则是一种无指导的死记式学习算法。

离散型霍普菲尔德网络的学习目的，是对具有 q 个不同的输入样本组 $P_{r\times q}=[P^1P^2\ \cdots\ P^q]$，希望通过调节计算有限的权值矩阵 W，使得当每一组输入样本 P^k, k=1, 2, …, q，作为系统的初始值，经过网络的工作运行后，系统能够收敛到各自输入样本矢量本身，当 $k=1$ 时，对于第 i 个神经元，由海布学习规则可得网络权值对输入矢量的学习关系式为：

$$w_{ij}=\alpha\, p_j^1 p_i^1 \tag{5.2}$$

其中，$\alpha>0$, $i=1, 2, \ldots, r$; $j=1, 2, \ldots, r$。在实际学习规则的运用中，一般取 $\alpha=1$ 或 $\alpha=\dfrac{1}{r}$。(5.2)式表明了海布调节规则：神经元输入 P 与输出 A 的状态相同（即同时为正或为负）时，从第 j 个到第 i 个神经元之间的连接强度 w_{ij} 则增强(即为正)，否则 w_{ij} 则减弱（为负）。

那么由(5.2)式求出的权值 w_{ij} 是否能够保证 $a_i=p_j$？取 $\alpha=1$，我们来验证一下，对于第 i 个输出节点，有：

$$a_i^1=\text{sgn}\Big(\sum_{j=1}^{r} w_{ij}p_j^1\Big)=\text{sgn}\Big(\sum_{j=1}^{r} p_j^1 p_i^1 p_j^1\Big)=\text{sgn}(p_i^1)=p_i^1$$

因为 p_i 和 a_i 值均取二值{-1, 1}，所以当其为正值时，即为 1；其值为负值时，即为-1。同符号值相乘时，输出必为 1。而且由 sgn(p_i^1)可以看出，不一定需要 sgn(p_i^1)的值，只要符号函数 sgn(·) 中的变量符号与 p_i^1 的符号相同，即能保证 sgn(·)= p_i^1。这个符号相同的范围就是一个稳定域。

当 $k=1$ 时，海布规则能够保证 $a_i^1=p_i^1$ 成立，使网络收敛到自己。现在的问题是：对于同一权矢量 W，网络不仅要能够使一组输入状态收敛到其稳态值，而且是要能够同时记忆住多个稳态值，即同一个网络权矢量必须能够记忆住多组输入样本，使其同时收敛到不同对应的稳态值。所以，根据海布规则的权值设计方法，当 k 由 1 增加到 2，直至 q 时，则是在原有已设计出的权值的基础上，增加一个新量 $p_j^k p_i^k$，$k=2, \ldots, q$，所以对网络所有输入样本记忆权值的设计公式为：

$$w_{ij}=\alpha\sum_{k=1}^{q} t_j^k t_i^k \tag{5.3}$$

式中矢量 T 为记忆样本，$T=P$。上式称为推广的学习调节规则。当系数 $\alpha=1$ 时，称(5.3)式为 T 的外积和公式。

DHNN 的设计目的是使任意输入矢量经过网络循环最终收敛到网络所记忆的某个样本上。

因为霍普菲尔德网络有 $w_{ij} = w_{ji}$，所以完整的霍普菲尔德网络权值设计公式应当为:

$$w_{ij} = \alpha \sum_{\substack{k=1 \\ i \neq j}}^{q} t_j^k t_i^k \tag{5.4}$$

用向量形式表示为:

$$W = \alpha \sum_{k=1}^{q} [T^k (T^k) T - I] \tag{5.5}$$

当 $\alpha = 1$ 时有:

$$W = \sum_{k=1}^{q} T^k (T^k)^T - qI \tag{5.6}$$

其中，I 为单位对角矩阵。

由(5.5)和(5.6)式所形成的网络权值矩阵为零对角阵。

采用海布学习规则来设计记忆权值，是因为设计简单，并可以满足 $w_{ij} = w_{ji}$ 的对称条件。从而可以保证网络在异步工作时收敛。在同步工作时，网络或收敛或出现极限环为 2 。在设计网络权值时，与前向网络不同的是令初始权值 $w_{ij} = 0$，每当一个样本出现时，都在原权值上加上一个修正量，即 $w_{ij} = w_{ij} + t_j^k t_i^k$，对于第 k 个样本，当第 i 个神经元输出与第 j 个神经元输入同时兴奋或同时抑制时，$t_j^k t_i^k > 0$；当 $t_j^k t_i^k$ 中一个兴奋一个抑制时，$t_j^k t_i^k < 0$。这就和海布提出的生物神经细胞之间的作用规律相同。

在神经网络工具箱中有关采用海布公式求解网络权矩阵变化的函数为 **learnh.m** 和 **learnhd.m**, 后者为带有衰减学习速率的函数:

dW = learnh (P, A, lr);

或 dW = learnhd (W, P, A, lr, dr);

对于简单的情况，lr 可以选择 1；对于复杂的应用，可取 lr = 0.1 ~ 0.5, dr = lr/3 。

下面给出 DHNN 在联想记忆上的应用例子。

【例 5.1】模式的记忆与联想。

用海布学习法则对下列模式学习记忆问题设计 DHNN 网络，并考察其联想性能。

$$P = T = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

解:

由题意可得记忆矢量为:

$$T = [T^1 \quad T^2 \quad T^3] = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

根据海布学习法则的外积和 DHNN 权值设计公式:

$$
\begin{aligned}
W &= \sum_{k=1}^{3}[T^k(T^k)^T - I] \\
&= T^1T^{1T} + T^2T^{2T} + ^3T^{3T} - 3I \\
&= \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -3 \\ -1 & -3 & 0 \end{bmatrix}
\end{aligned}
$$

验证:

将目标矢量作为输入矢量带人网络中可得:

$$A^1 = \mathrm{sgn}(WP^1) = T^1$$

$$A^2 = \mathrm{sgn}(WP^2) = T^2$$

$$A^3 = \mathrm{sgn}(WP^3) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} = T^2 \quad T^3$$

由此可见，采用海布学习规则设计出的离散型霍普菲尔德记忆网络，并没有准确的记忆所有期望的模式。这个情况是如何发生的呢?

5.3.4 影响记忆容量的因素

设计 DHNN 网络的目的，是希望通过所设计的权值矩阵 W 储存多个期望模式。从海布学习公式的推导过程中可以看出：当网络只记忆一个稳定模式时，该模式肯定被网络准确无误地记忆住，即所设计的 W 值一定能够满足正比于输入和输出矢量的乘积关系。但当需要记忆的模式增多时，情况则发生了变化，主要表现在下面两点上:

（1）权值移动

在网络的学习过程中，网络对记忆样本输入 $T^1, T^2, \ldots, T^q$ 的权值学习记忆实际上是逐个实现的。即对权值 W，有程序:

$$
\begin{aligned}
&W = 0 \\
&\text{for} \quad k = 1, q \\
&W = W + T^kT^{kT} - I \\
&\text{end}
\end{aligned}
$$

由此过程可知：当 $k = 1$ 时，有

$$w_{ij} = t_j^1 t_i^1 \quad (i \neq j)$$

$$a_i^1 = \mathrm{sgn}(\sum_{j=1}^{r} w_{ij} t_j^1) = \mathrm{sgn}(\sum_{j=1}^{r} t_j^1 t_i^1 t_j^1) = t_i^1$$

此时，网络准确的记住了样本 T^1，当 $k = 2$ 时，为了记忆样本 T^2，需要在记忆了样本 T^1 的权值上加上对样本 T^2 的记忆项 $T^2T^{2T} - I$。将权值在原来值的基础上产生了移动。在此情况下，所求出的新的 W 为：$w_{ij} = t_j^1 t_i^1 + t_j^2 t_i^2$，对于样本 T^1 来说，网络的输出为:

$$a_i^1 = \mathrm{sgn}(\sum_{j=1}^{r} w_{ij} t_j^1) = \mathrm{sgn}[t_i^1 + \sum_{j=1}^{r}(t_j^2 t_i^2 t_j^1)]$$

此输出有可能不再对所有的 s 个输出均满足加权输入和与输出符号一致的条件。网络

有可能部分地遗忘了以前已记忆住的模式。

另一方面，由于在学习样本 T^2 时，权矩阵 W 是在已学习了 T^1 的基础上进行修正的。此时，因 W 起始值不再为零，所以由此调整得出的新的 W 值，对记忆样本 T^2 来说，也未必对所有的 s 个输出同时满足符号函数的条件，即难以保证网络对 T^2 的精确的记忆。

随着学习样本数 k 的增加，权值移动现象将进一步发生，当学习了第 q 个样本 T^q 后，权值又在前 q–1 个样本修正的基础上产生了移动，这也是网络在精确的学习了第一个样本后的第 q–1 次移动。不难想象，此时的权矩阵 W 对于 P^1 来说，使每个输出继续能够同时满足符号条件的可能性有多大？同样，对于其他模式 T^k，$k=2, ..., q-1$，也存在着同样的问题。很有可能出现的问题是：网络部分甚至全部地遗忘了先前已学习过的样本。即使对于刚刚进入网络的样本 T^q，由于前 q–1 个样本所形成的记忆权值难以预先保证其可靠性，因而也无法保证修正后所得到的最终权值矩阵 W 满足其符号条件。这也就无法保证网络能够记忆住该样本 T^q。

从动力学的角度来看，k 值较小时，网络的海布学习法则，可以使输入学习样本成为其吸引子。随着 k 值的增大，不但难以使后来的样本成为网络的吸引子，而且有可能使已记忆住的吸引子的吸引域变小，使原来处于吸引子位置上的样本从吸引子的位置发生移动。对已记忆的样本发生遗忘，这种现象成为“疲劳”。

（2）交叉干扰

设网络的权矩阵已经设计完成，网络的输入矢量为 P，并希望其成为网络的稳定矢量 $T=P$，按同步更新规则，状态演变方程为：

$$A=P=\mathrm{sgn}(N)=\mathrm{sgn}(WP)$$

实际上，上式就是 P 成为稳定矢量的条件，式中 N 为神经网络的加权输入和矢量。

设输入矢量 P 维数为 $r\times q$,取$\alpha=\frac{1}{r}$，因为对于 DHNN 有 $P^k\in\{-1, 1\}, =1$， $2,\dots, q$，所以有 $p_i^k\cdot p_i^k=p_j^kp_j^k=1$。当网络某个矢量 P^l，$l\in[1, q]$，作为网络的输入矢量时，可得网络的加权输入和 n_i^l 为：

$$\begin{aligned} n_i^l &= \sum_{\substack{j=1\\ j\neq i}}^{r} w_{ij}p_j^l \\ &= \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{r}\sum_{k=1}^{q} p_i^k p_j^k \;\; p_j^l \\ &= \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{r}\left[p_i^l\cdot p_j^l\cdot p_j^l+\sum_{\substack{k=1\\ k\neq l}}^{q} p_i^k p_j^k\cdot p_j^l\right] \\ &= p_i^l+\frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{r}\sum_{\substack{k=1\\ k\neq l}}^{q} p_i^k p_j^k\cdot p_j^l \end{aligned} \tag{5.7}$$

上式右边中第一项为期望记忆的样本，而第二项则是当网络学习多个样本时，在回忆阶段即验证该记忆样本时，所产生的相互干扰，称为交叉干扰项。由 sgn 函数的符号性质可知，(5.7)式中第一项可使网络产生正确的输出，而第二项可能对第一项造成扰动，网络

对于所学习过的某个样本能否正确的回忆，完全取决于(5.7)式中第一项与第二项的符号关系及数值大小。

下面我们来分析(5.7)式中第二项对网络的影响。令

$$C_i^l = p_i^l \Big/ \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{r}\sum_{\substack{k=1\\ k\neq l}}^{q} p_i^k p_j^k \cdot p_j^l \tag{5.8}$$

1）$C_i^l \geqslant 0$：

此时，交叉干扰项与 p_i^l 符号一致，因此，能够在网络的输出端得到正确的结果，即有 $a_i^l = p_i^l$；

2）$C_i^l \leqslant -1$：

此时，交叉干扰项与 p_i^l 符号相反，对正确记忆出 p_i^l 造成不利影响，但因为干扰幅度较小，而不至于使状态 n_i^l 的符号翻转。因而输出仍然正确，可得到 $a_i^l = p_i^l$；

3）$-1 < C_i^l < 0$：

此时，交叉干扰项与 p_i^l 不仅符号相反，而且其值幅度大于正确的输出信号值 p_i^l，从而造成状态 n_i^l 的符号翻转，导致网络的第 i 个神经元输出错误的信息：$a_i^l = -p_i^l$。

5.3.5 网络的记忆容量确定

从对网络的记忆容量产生影响的权值移动和交叉干扰上看，采用海布学习法则对网络记忆样本的数量是有限制的，通过上面的分析已经很清楚地得知，当交叉干扰项幅值大于正确记忆值时，将产生错误输出，那么，在什么情况下，能够保证记忆住所有样本？答案是有的。当所期望记忆的样本是两两正交时，能够准确得到一个可记忆数量的上限值。

在神经元为二值输出的情况下，即 $P_j \in \{-1,1\}$，当两个 r 维样本矢量的各个分量中，有 $\frac{r}{2}$ 是相同的 1，有 $\frac{r}{2}$ 是相反的-1，对于任意一个数 l，$l \in [1,r]$，有 $P^l(P^k)^T = 0$，$l \neq k$；而有 $P^l\ (P^l\ )^T=1, l = k$ 的正交特性。下面用外积和公式所得到的权矩阵进行迭代计算，在输入样本 $P^k, k=1,2\ \dots,q$ 中任取一个 P^l 作为初始输入，求网络的加权输入和 N^l：

$$N^l = WP^l = [P^1P^2\dots P^l\dots P^q]\begin{bmatrix} P^{1T}\\ P^{2T}\\ \vdots\\ P^{lT}\\ \vdots\\ P^{nT}\end{bmatrix} P^l - qP^l$$

$$= [P^1P^2\dots P^l\dots P^q]\begin{bmatrix} 0\\ 0\\ \vdots\\ P^{lT}\ P^l\\ \vdots\\ 0\end{bmatrix} - qP^l$$

$$= P^l P^{lT} P^l - qP^l$$
$$= (r - q)P^l$$

由上式结果可知，只要满足 $r > q$ ，则有 $\text{sgn}(N^l) = P^l$ ，保证 P^l 为网络的稳定解。

但对于一般的非正交的记忆样本，从前面的交叉干扰的分析过程中已经得知，网络不能保证收敛到所希望的记忆样本上。

【例 5.2】采用外积和记忆公式设计 DHNN 的权值，希望记忆的样本为：

$$T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ 1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} = [T^1 \quad T^2 \quad T^3]$$

解:

由外积和权值设计公式，可得权矩阵为：

$$W = \sum_{k=1}^{3} T^k T^{kT} - 3I$$
$$= \begin{bmatrix} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{bmatrix}$$

经验证所设计的网络能够准确的回忆出记忆的全部三个模式。实际上，该网络可能的输入状态有 $2^r = 2^5 = 32$ 种，分析一下其稳定点的情况可得，系统共有四个稳定点，分别为：

$$T = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = [T^1 \quad T^2 \quad T^3 \quad T^4]$$

四个稳定点都是渐近稳定点，其中 T^1，T^2 和 T^3 为要求的稳定点，即网络的解。而 T^4 为伪稳定点有 $T^4 = -T^2$。在用串行方式工作的情况下，在 32 种输入状态中，有 8 个初始态收敛到 T^1， 9 个初始态收敛到 T^2， 5 个初始态收敛到 T^3， 10 个初始态收敛到 T^4。在用并行方式工作的情况下，有 10 个初始态收敛到 T^1， 1 个初始态收敛到 T^2， 2 个初始态收敛到 T^3， 1 个初始态收敛到 T^4，而有 18 个初始态均陷入到极限环，其收敛域明显减少。

DHNN 用于联想记忆有两个突出的特点：即记忆是分布式的，而联想是动态的。这与人脑的联想记忆实现机理相类似。利用网络稳定的平衡点来存储记忆样本，按照反馈动力学运动规律唤起记忆，显示了 DHNN 联想记忆实现方法的重要价值。然而， DHNN 也存在有其局限性，主要表现在以下几点：①记忆容量的有限性；②伪稳定点的联想与记忆；③当记忆样本较接近时，网络不能始终回忆出正确的记忆等。

另外网络的平衡稳定点并不可以任意设置的。也没有一个通用的方式来事先知道平衡稳定点。换句话说，并没有一个简便的方法来求网络的平衡稳定点。只有靠用一个一个样本去测试寻找。所以真正想利用好霍普菲尔德网络并不是一件容易的事情。

5.3.6 DHNN 权值设计的其他方法

用海布规则设计出的 DHNN 权值能够保证其网络在异步工作时稳定收敛，尤其在记忆样本是正交的条件下，可以保证每个记忆样本收敛到自己，并有一定范围的吸引域，但对于那些不是正交的记忆样本，用此规则设计出的网络则不一定能收敛到本身。

这里介绍几种其他的权值设计方法以对此不足加以改进，它们都各有自己的特点。

（1） *δ学习规则*

δ学习规则基本公式为:

$$\Delta W = \eta \cdot \delta \cdot P,$$
$$w_{ij}(t+1) = w_{ij}(t) + \eta[T(t) - A(t)]P(t)$$

即通过计算每个神经元节点的实际激活值 $A(t)$，与期望状态 $T(t)$进行比较，若不满足要求，则将二者的误差的一部分作为调整量，若满足要求，则相应的权值保持不变。

（2） *伪逆法*

对于输入样本 $P=[P^1 \quad P^2 \quad \dots \quad P^q]$，设网络输出可以写成一个与输入样本相对应的矩阵 A，输入和输出之间可用一个权矩阵 W 来映射，即有: $W*P = N, A = \text{sgn}(N)$，由此可得:

$$W=N*P* \tag{5.9}$$

其中 $P*$为 P 的伪逆，有 $P*=(P^TP)^{-1}P^T$，如果样本之间是线性无关的，则 P^TP 满秩，其逆存在，则可求出(5.9)式求权矩阵 W 来。

用伪逆法求出的权矩阵 W，可以保证对所记忆的模式，在输入时仍能够正确收敛到样本自己，在选择 A 值时，只要满足 A 矩阵中的每一个元素与 $W*P$ 矩阵中的每个元素有相同的符号，甚至可以简单地选择 A 与 P 具有相同符号的值，即可满足收敛到学习样本的本身。但当记忆样本之间是线性相关的，如【例 5.1 】中有 $T^2 = -T^1$，以及对【例 5.2 】中有 $T^4 = -T^2$，对于这些情况，由海布法所设计出网络存在的问题，伪逆法也解决不了，甚至无法求解，相比之下，由于存在求逆等运算，伪逆法较为繁琐，而海布法则要容易求得多。

【例 5.3 】采用伪逆法解【例 5.2 】。其期望矢量为:

$$T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ 1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} = [T^1 \quad T^2 \quad T^3]$$

解:

取与输入矢量相同符号的元素作为 A 的值:

$$A=\begin{bmatrix} 0.5 & 0.5 & -0.1 \\ 0.5 & -0.1 & 0.5 \\ 0.5 & -0.1 & 0.5 \\ 0.5 & 0.5 & -0.1 \\ 0.1 & -0.5 & -0.5 \end{bmatrix}$$

则:

$$w = A*P*=A(P^TP)^{-1}P^T$$

$$=\begin{bmatrix} 0.25 & 0.1 & 0.1 & 0.25 & -0.2 \\ 0.1 & 0.25 & 0.25 & 0.1 & -0.2 \\ 0.1 & 0.25 & 0.25 & 0.1 & -0.2 \\ 0.25 & 0.1 & 0.1 & 0.25 & -0.2 \\ -0.1 & -0.1 & -0.1 & -0.1 & 0.5 \end{bmatrix}$$

此 W 可以保证输入 T^1， T^2 和 T^3 收敛到自己。

（3）正交化的权值设计

这一方法的基本思想和出发点是为了满足下面四个要求:

1) 保证系统在异步工作时的稳定性，即它的权值是对称的，满足 $w_{ij}=w_{ji}$, $i, j = 1, 2, ..., s$;

2) 保证所有要求记忆的稳定平衡点都能收敛到自己;

3) 使伪稳定点的数目尽可能的少;

4) 使稳定点的吸引域尽可能的大。

正交化权值计算公式推导如下:

a)已知有 q 个需要存储的稳定平衡点 $T^1, T^2, ..., T^q, T \in R^s$，计算 $s\times(q-1)$ 阶矩阵 $YR^{s\times(q-1)}$:

$$Y = [\, T^1 - T^q \quad T^2 - T^q \quad \cdots \quad T^{q-1} - T^q \,]^T$$

b)对 Y 进行奇异值及酋矩阵分解，如存在两个正交矩阵 U 和 V 以及一个对角值为 Y 的奇异值的对角矩阵 A，满足:

$$Y = UAV$$

$$Y = [\, T^1\, T^2 \dots T^{q-1} \,]^T$$

$$U = \left[U^1U^2 \dots U^s\right]^T$$

$$V = \left[V^1V^2 \dots V^{q-1}\right]^T$$

$$A = \begin{bmatrix} \lambda_1 & \cdots & & 0 \\ & \ddots & & \\ \vdots & & \lambda_k & \vdots \\ & & & \ddots \\ 0 & & \cdots & 0 \end{bmatrix}$$

k 维空间为 s 维空间的子空间，它由 k 个独立基组织:

$$k = \mathrm{rank}\,(A)$$

设 $\{U^1U^2 \dots U^k\}$ 为 Y 的正交基，而 $\{U^{k+1}U^{k+2} \dots U^s\}$ 为 s 维空间中的补充正交基，下

面利用 U 矩阵来设计权值。

c)定义:

$$W^+ = \sum_{j=1}^{k} U^i (U^i)^T, \quad W^- = \sum_{j=k+1}^{s} U^i (U^i)^T$$

总的联接权值为:

$$W_\tau = W^+ - \tau W^-$$

其中τ 为大于-1 的参数。

d)网络的阀值定义为:

$$B_\tau = T^q - W_\tau T^q$$

由此可见，网络的权矩阵是由两部分的权矩阵W^+和W^- 相加而成的，每一部分权所采用的都是类似于外积和法得到的，只是用的不是原始要求记忆的样本，而是分解后正交矩阵的分量。这两部分权矩阵均满足对称条件，即有下式成立:

$$w_{ij}^+ = w_{ji}^+; \quad w_{ij}^- = w_{ji}^-$$

因而$W\tau$ 中分量也满足对称条件。这就保证了系统在异步时能够收敛并且不会出现极限环。

下面我们来推导记忆样本能够收敛到自己的有效性。

①对于输入样本中的任意目标矢量T^i，$i = 1, 2, ..., q$，因为($T^i - T^q$)是 Y 中的一个矢量，它属于 A 的秩所定义的 k 个基空间中的矢量，所以必存在一些系数$\alpha 1, \alpha 2, ..., \alpha k$ 使

$$T^i - T^q = \alpha 1 U^1 + \alpha 2 U^2 + ... + \alpha k U^k$$

即:

$$T^i = \alpha 1 U^1 + \alpha 2 U^2 + ... + \alpha k U^k + T^q$$

对于 U 中任意一个U^i，有:

$$W_\tau U^i = W^+ U^i - \tau W^- U^i = U^i$$

对于样本输入 T^i，其网络输出为:

$$\begin{aligned} A^i &= \mathrm{sgn}\left(W_\tau T^i + B_\tau\right) \\ &= \mathrm{sgn}\left(W^+ T^i - \tau W^- T^i + T^q - W^+ T^q + \tau W^- T^q\right) \\ &= \mathrm{sgn}\left[W^+ (T^i - T^q) - \tau W^- (T^i - T^q) + T^q\right] \\ &= \mathrm{sgn}\left[(T^i - T^q) + T^q\right] \\ &= T^i \end{aligned}$$

②当选择第 q 个样本 T^q 作为输入时，有:

$$\begin{aligned} A^q &= \mathrm{sgn}\left(W_\tau T^q + B_\tau\right) \\ &= \mathrm{sgn}\left(W_\tau T^q + T^q - W_\tau T^q\right) \\ &= \mathrm{sgn}\left(T^q\right) \\ &= T^q \end{aligned}$$

③如果输入一个不是记忆样本的 P，则网络输出为:

$$A = \mathrm{sgn}(W_\tau P + B_\tau) = \mathrm{sgn}\left[\left(W^+ - \tau W^-\right)\left(P - T^q\right) + T^q\right]$$

因为 P 不是已学习过的记忆样本，$P - T^q$ 不是 Y 中的矢量，则必然有 $W_\tau(P-T^q) \neq P-T^q$，并且在设计过程中可以通过调节 $W_\tau = W^+ - \tau W^-$ 中的参数τ 的大小，来控制（$P - T^q$）与 T^q 的符号，以保证输入矢量 P 与记忆样本之间存在足够的大小余额，从而使 $\mathrm{sgn}(W_\tau P + B_\tau) \neq P$。使 P 不能收敛到自身。

利用参数τ 的调节可以改变伪稳定点的数目。在串行工作的情况下，伪稳定点数目的减少就意味着每个期望稳定点的稳定域的扩大。对于任意一个不在记忆中的样本 P，总可以设计一个τ 把 P 排除在外。

表 5.1 给出的是一个$\tau = 10$，学习记忆 5 个稳定样本的系统，采用上面的方法进行权的设计，以及在不同的τ 时的稳定点数目。同时给出了正交化设计方法与外积和网络权值设计法的比较。

表 5.1　不同的τ 时的稳定点数目

	$\tau = 1$	$\tau = 10$	外积和公式设计
稳定点数	8	5	4
错误稳定点数	0	0	5

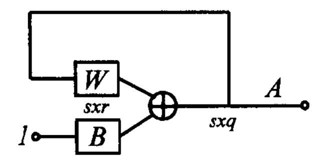
虽然正交化设计方法的数学设计较为复杂，但与外积和法相比较，所设计出的平衡稳定点能够保证收敛到自己并且有较大的稳定域。更主要的是在 MATLAB 工具箱中已将此设计方法写进了函数 **solvehop.m** 中：

```
[ W, b ] = solvehop(T);
```

用目标矢量给出一组目标平衡点，由函数 **solvehop.m** 可以设计出对应反馈网络的权值和偏差，保证网络对给定的目标矢量的输入能收敛到稳定的平衡点。但网络可能也包括其他伪平衡点，这些不希望点的数目通过选择τ 值（缺省值为 10）已经做了尽可能的限制。

一旦设计好网络，可以用一个或多个输入矢量对其进行测试。这些输入将趋近目标平衡点，最终找到它们的目标矢量。下面给出用正交化方法设计权值的例子。

【例 5.4】考虑一个具有两个神经元的霍普菲尔德网络，每个神经元具有两个权值和一个偏差。



A = satlin(W*P, B);

图 5.7　正交化方法设计的霍普菲尔德网络结构图

网络所要存储的目标平衡点为一个列矢量 T：

```
T = [ 1 -1;
     -1 1 ];
```

T 将被用在设计函数 **solvehop.m** 中以求出网络的权值：

```
[ W, b ] = solvehop(T);
```

solvehop.m 函数返回网络的权值与偏差为:

```
W = [ 0.6925 -0.4694;
      -0.4694 0.6925 ]
b = 1.0 e-16 * [0.6900; 0.6900]
```

可以看出其权值是对称的。

下面用目标矢量作为网络输入来测试其是否已被存储到网络中了。取输入为:

```
Ptest = [ 1 -1; -1 1 ];                % 输入矢量
```

用来进行测试的函数为 **simhop.m**；

```
A = simhop(Ptest, W, B, 3);            % 计算其网络循环结束后的输出
```

函数右端中的第四个参数 3 表示网络反复循环的次数。经过 3 次的运行，如同所希望的那样，网络的结果为目标矢量 T。

现在我们想知道所设计的网络对任意输入矢量的收敛结果。我们首先给出六组输入矢量为:

```
P = [ 0.5621   0.3577   0.8694   0.0388   -0.9309   0.0594;
     -0.9059   0.3586  -0.2330   0.6619    0.8931   0.3423 ];
```

在经过 25 次循环运行后，得到输出为:

```
A = [ 1.0000   -0.0191   1.0000   -1.0000   -0.8036   -1.0000;
     -1.0000    0.0191  -1.0000    1.0000    0.8036    1.0000 ];
```

如同所看到的第 2 组及第 5 组的输入矢量没有能够收敛到目标平衡点上。然而，如果让其运行 60 次，它们则能够收敛到第 2 个目标矢量上。

另外，当取其他随机初始值时，在 25 次循环后均能够收敛到所设计的平衡点上。一般情况下，此网络对任意初始随机值，在运行 60 次左右均能够收敛到网络所储存的某个平衡点上。

【例 5.5】观察不稳定平衡点的位置。

同样采用【例 5.4】的网络，使网络存储的平衡点为:

$$T = [\ 1\ -1;\ -1\ 1\ ];$$

由【例 5.4】已知，当我们用任意随机输入矢量去对网络进行测试时，在 60 次循环以内均能够使其收敛到所设计的平衡点上，如图 5.8 所示。图中任意随机输入矢量 “+”最终都收敛到网络的稳定平衡点 T “*” 上。

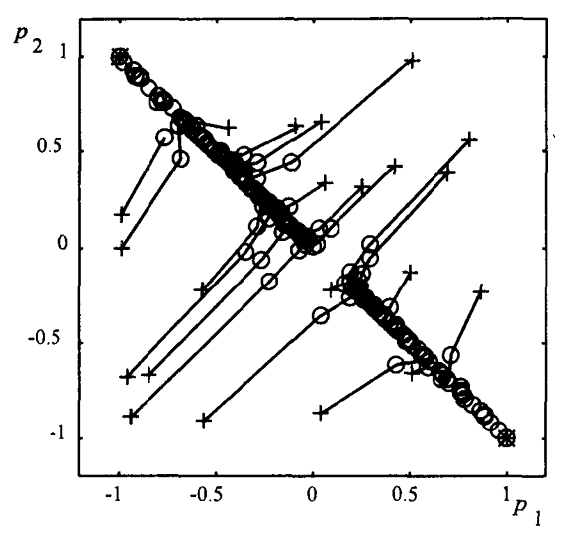


图 5.8 网络任意输入矢量的收敛过程

然而这些并不能保证网络没有其他平衡点了，让我们来尝试一下。例如，取所给目标矢量连线的垂直平分线上的点作为输入矢量:

$$P = [\,-1\ -0.5\ \ 0\ \ 0.5\ \ 1.0;\ \ -1\ -0.5\ \ 0\ \ 0.5\ \ 1.0\,];$$

首先采用与【例 5.4】相同的步骤设计出网络。然后将上述 *P* 作为网络输入，采用下面的循环程序来观察每一次循环后网络的输出值:

```
for k = 1:4
disp('Hit<RETURN> to see the output after one cycle.');
pause, disp('')
A = simhop(P,W,B,k);
fprintf('Output Vectors:'),
A
end
```

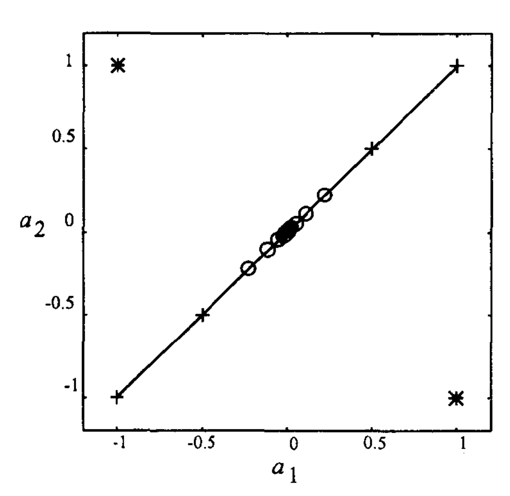


图 5.9 不稳定平衡点的收敛走向

通过循环可以看到所有的输出最终趋于原点。它是一个不稳定的平衡点，因为任何一个不在以(-1, -1) 到(1, 1)点所连成的直线上的点作为网络的初始输人矢量，经过循环都将最终收敛到所设计的稳定平衡点上。网络输出平面上的点的收敛走向如图 5.9 所示。

【例 5.6】设计一个三元的霍普菲尔德网络，使网络存储的目标平衡点为:

```
T = [ 1   1;
     -1   1;
     -1  -1];
```

同样只要采用函数 **solvehop.m** 即可求出所要设计网络的权值为:

```
W = [ 0.2231      0       0 ;
        0       11618     0;
        0         0     02231 ];
B = [ 0.8546; 0; -0.8546 ];
```

通过使目标矢量作为输入矢量，用函数 **simhop.m** 运行网络，可以看到网络确实能够使其收敛到自己本身。同样的问题是，网络是否还有其他的平衡点：可以通过取任意初始值的方式对其进行测试。取循环次数为 50，其结果是：一般情况下总能够收敛到某个目标矢量上，如图 5.10 所示。

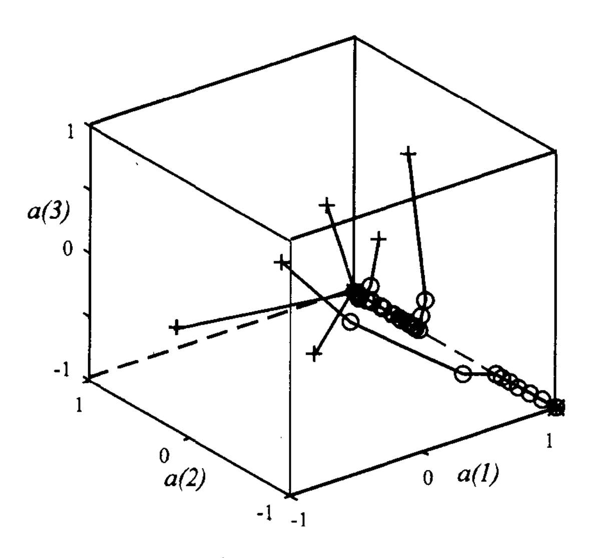


图 5.10　三维空间的输出及其状态收敛趋势

然后，再尝试用两目标矢量点连线的平分面上的点作为初始矢量，如:

```
P = [ 1  -1   -0.5     1      1      0 ;
      0   0    0       0     0.01   -0.2;
     -1   1    0.5   -1.01   -1      0 ];
```

网络经过 5 次循环后得输出为:

```
A = [ 1   1    1   1     1      1;
```

0 0 0 0 0.0212 -0.4234;
-1 -1 -1 -1 -1 -1];

前四个矢量全部收敛到不稳定的平衡点(1 0 -1)上。然而，既使第五和第六个矢量起始于非常接近于不稳定的平衡点，它们仍然能够以自己的方式逐步趋于所设计的平衡点。由此可见，非常简单的例子也存在不稳定的平衡点。

霍普菲尔德网络三维空间的输出及其状态收敛趋势如图 5.11 所示。

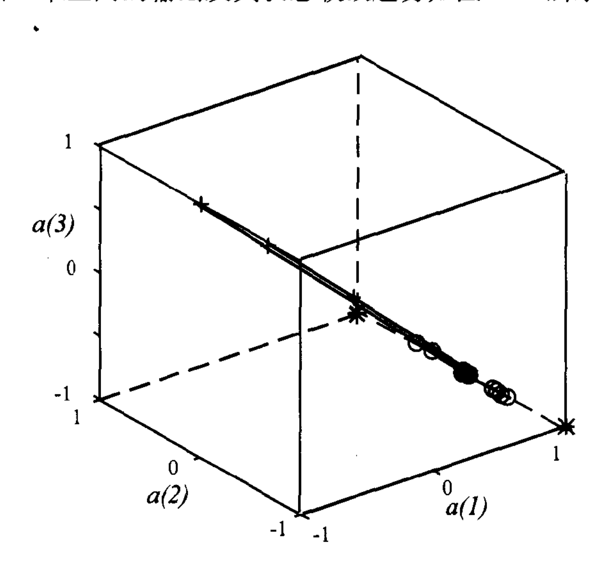


图 5.11 三维空间不稳定的平衡点

【例 5.7 】伪稳定平衡点。

设计一个存储下面 4 组 5 神经元平衡点的神经网络：

```
T = [ 1   1  -1   1;
     -1   1   1  -1;
     -1  -1  -1   1;
      1   1   1   1;
     -1  -1   1   1 ];
```

执行程序：

```
[W,B] = solvehop(T);
```

得网络设计结果为：

```
W = [ 0.8489  -0.0    0.3129   0.0    -0.3129;
     -0.0     1.1618  0        0      -0.0;
      0.3129  0.0     0.8489   0       0.3129;
      0       0       0        0.223   0;
```

```
     -0.3129    -0.0      0.3129      0        0.8489];
B = [ 0.2849;      0 ;     -0.2849;   0.8546;   0.2849 ];
```

通过检验程序:

```
A = simhop(T,W,B,3),
```

可得到证实它们确实能够收敛到自己本身。

然后我们向网络输入随机初始矢量。结论是，偶尔有些状态稳定到伪渐近稳定点上，如下面所示结果中的第二列和第三列，此网络除了具有所设计的稳定平衡点外，还由另外两个伪稳定点:

```
A = [ -1   1   -1   1   1   -1;
       1   1   -1  -1  -1    1;
      -1   1   -1  -1   1   -1;
       1   1    1   1   1    1;
       1   1    1  -1   1    1 ];
```

5.4 连续型霍普菲尔德网络

霍普菲尔德网络可以推广到输入和输出都取连续数值的情形。这时网络的基本结构不变，状态输出方程形式上也相同。若定义网络中第 i 个神经元的输入总和为 n_i，输出状态为 a_i，则网络的状态转移方程可写为:

$$a_i = f(\sum_{j=1}^{r} w_{ij} p_j + b_i)$$

其中神经元的激活函数 f 为 S 型的函数（或线性饱和函数）:

$$f1 = \frac{1}{1+e^{-\lambda(n_i+b_i)}}$$

或

$$f2 = \tanh(\lambda(n_i + b_i))$$

两个函数共同的特点当 $n_i \to \infty$ 及 $n_i \to -\infty$ 时，函数值饱和于两极，从而限制了神经网络中输出状态 a_i 的增长范围。显然，若使用函数 $f1(\cdot)$，则 $a_i \in [0,1]$，若使用 $f2(\cdot)$，则 $a_i \in [-1,1]$。函数 $f1(\cdot)$ 和 $f2(\cdot)$ 中的参数 λ 用以控制 S 型函数在 0 点附近的变化数。

在连续网络的整个运行过程中，所有神经元状态的改变具有三种形式: 异步更新、同步更新和连续更新。与离散的网络相比，连续更新是一种新的方式，表示网络所有神经元都随连续时间 t 并行更新。就像用电子元件实现的霍普菲尔德网络神经元状态随电路参量改变一样。下面将要讲到网络模型及运行过程，可以很清楚地理解这一点。另外，离散霍普菲尔德网络中状态改变着网络输出在“1”与“-1”之间翻转，而这里，网络状态是在

一定范围内的连续变化。

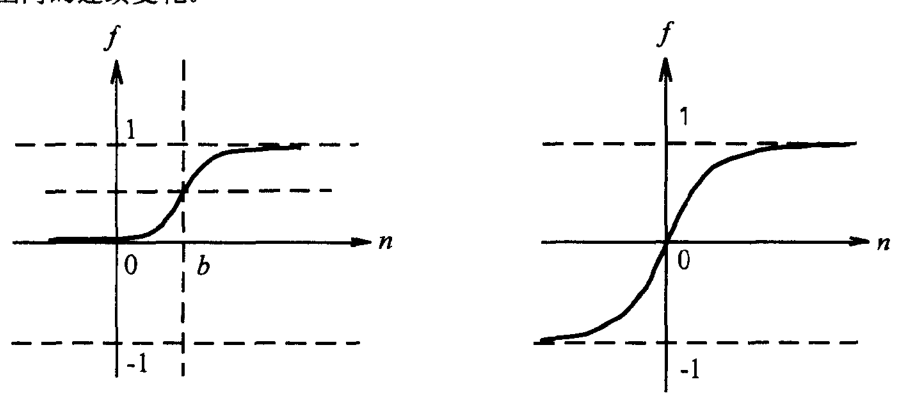


图 5.12　连续霍普菲尔德网络激活函数

5.4.1　对应于电子电路的网络结构

电子电路与 CHNN 之间存在着直接的对应关系。第 i 个输出神经元的模型如图 5.13 所示，其中，运算放大器模拟神经元的激活函数，电压 u_i 为激活函数的输入，也称为网络的状态，各并联的电阻 R_{ij} 值决定各神经元之间的连接强度；电容 C 和电阻 r 模拟生物神经元的输出时间常数，另外电流 I_i 模拟阀值。其中， $i,j=1,2,...,r$ 。整个网络是由 r 个同样的模型并联组成，每个模型都具有相同的输入矢量 $V=[v_1,v_2,...,v_r]$，状态矢量和 u_i 由每个模型的输出 v_i 组合而成。电路状态与输出之间的关系图由图 5.14 所示。

图 5.14 也是放大器的特性图。

对于图 5.13 所示的电路图，根据克西霍夫电流定律，有下列方程成立:

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t}+\frac{u_i}{r}=\sum_{j=1}^{r}\frac{1}{R_{ij}}(v_j-u_i)+I_i$$

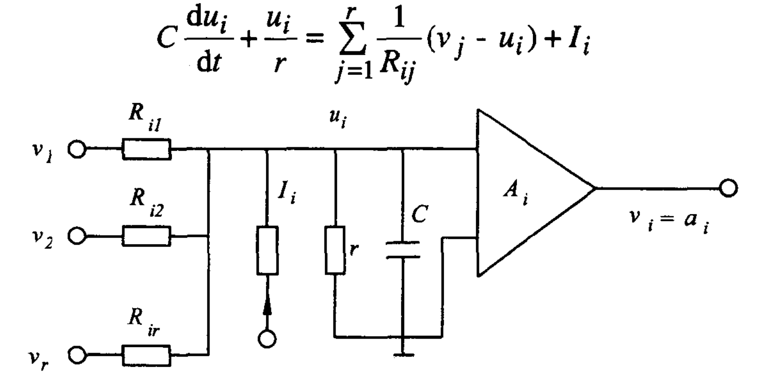


图 5.13　第 i 个输出神经元电路模式

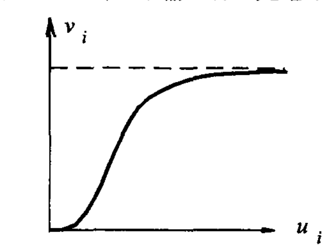


图 5.14　网络输出 v_i 与状态 u_i 的关系图

对上式进行移项及合并，可得：

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t}=-(\frac{u_i}{r}+\sum_{j=1}^{r}\frac{1}{R_{ij}})u_i+\sum_{j=1}^{r}\frac{1}{R_{ij}}v_j+I_i$$

若令 $\frac{1}{R}=\frac{1}{r}+\sum_{j=1}^{r}\frac{1}{R_{ij}}$， $w_{ij}=\frac{1}{R_{ij}}$，

则可得：

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t}=-\frac{1}{R}u_i+\sum_{j=1}^{r}w_{ij}v_i+I_i$$

由此可得电路输入的累加值为：

$$s_i=\sum_{=1}^{r}w_{ij}v_j+I_i \tag{5.10}$$

(5.10)式与人工神经网络模型的加权值一致。

电路状态值与加权输入值之间的关系可用一阶微分方程式表示：

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t}=\frac{1}{R}u_i+s_i \tag{5.11}$$

而电路输出与状态之间的关系为一个单调上升的有界函数：

$$v_i=f(u_i) \tag{5.12}$$

方程(5.11)式反映了网络状态连续更新的意义，它与离散形式是不同的。（ 5.10 ）~（ 5.12 ）式结合在一起描述了 CHNN 的动态过程随着时间的流逝，网络趋于稳定状态，可以在输出端得到稳定的输出矢量。对于由相互连接的电路模型组成的网络，每个模型均满足（ 5.10 ）~（ 5.12 ）方程，可将其写成矩阵形式，为了简便起见，令 $C=1$ 。

$$\begin{bmatrix}\dot{u}_1(t)\\ \dot{u}_2(t)\\ \vdots\\ \dot{u}_r(t)\end{bmatrix}=-\frac{1}{R}\begin{bmatrix}\dot{u}_1(t)\\ \dot{u}_2(t)\\ \vdots\\ \dot{u}_r(t)\end{bmatrix}+\begin{bmatrix}w_{11} & w_{12} & \cdots & w_{1r} & v_1(t)\\ w_{21} & w_{22} & \cdots & w_{2r} & v_2(t)\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ w_{r1} & w_{r2} & \cdots & w_{rr} & v_{r1}(t)\end{bmatrix}+\begin{bmatrix}I_1\\ I_2\\ \vdots\\ I_r\end{bmatrix}$$

或

$$\dot{U}=-\frac{1}{R}U+WV+I \tag{5.13}$$

这是一个 r 维的线性微分方程。当 $\dot{u}_i(t)=0, i=1,2,\ldots,r$ ，上式变为：

$$U=R(WV+I)$$

若把 $I=[I_1I_2\Lambda\ I_r]^T$ 看作阈值，那么上式就与人工神经网络的加权输入和 $N+B$ 形式相似。如果此时代表激活函数 $F(U)$的运算放大器的放大倍数足够大到可视为二值型的硬函数，连续型反馈网络即变为离散型的反馈网络。所以也可以说， DHNN 是 CHNN 的一个特例。

5.4.2　CHNN 方程的解及稳定性分析

对于 CHNN 来说，我们关心的同样是稳定性问题。由于与实际电子电路有明确的对应关系，所以可以通过选择合适的电路参数来设计出稳定的 CHNN 电路来，以达到一定的目的。在所有影响电路系统稳定的所有参数中，一个比较特殊的参数值是放大器的放大倍数。从前面的分析中我们已经知道当放大器的放大倍数足够大时，网络将由连续型转化为离散型，状态与输出之间的关系表现了激活函数的形状，而正是激活函数代表了一个网络的特点，所以，下面我们着重分析不同 $V=F(U)$之间的激活函数关系对系统的稳定性的影响。

1）当 $v_i=\beta u_i, i=1,2,\ldots,r$ 时:

即输出 v_i 与状态 u_i 之间呈线性关系时，此时，系统的状态方程，由（5.13）式可写成标准形式:

$$\dot{U}=AU+B \tag{5.14}$$

其中: $A=-\frac{1}{R}+W\beta$ 。

此系统的特征方程为:

$$|A-\lambda I|=0$$

其中 I 为单位对角矩阵。通过对解出的特征值 $\lambda_1,\lambda_2,\ldots,\lambda_r$ 的不同情况，可以得到以下几种系统解的情况:

a)若 $\lambda_1,\lambda_2,\ldots,\lambda_r$ 的实部均小于 0 ，则系统稳定，此时，若给定系统初始值，经过运行，系统的状态最终能够收敛到各自的稳定点上。

b) $\lambda_1,\lambda_2,\ldots,\lambda_r$ 中若出现异号同值实根（即相同实根），系统则出现鞍点，即系统状态随时间变化的轨迹从某些方向是向着平衡点靠近，而在另一些方向上是远离平衡点的，因而此时，系统不能稳定到稳定点上，如图 5.15(d)。

c) $\lambda_1,\lambda_2,\ldots,\lambda_r$ 中若有零实部，且有零虚部，则系统在其状态空间上出现极限环。

d) $\lambda_1,\lambda_2,\ldots,\lambda_r$ 中若有大于零的实部，则系统发散。在二维状态情况下，不同的特征根对应的状态轨迹如图 5.15(f)和(g)所示，令平衡点为 $u_1=u_2=0$;

2）对于 $v_i=F(u_i)$ 为非线性的情况时，此时状态方程(5.14)可写为:

$$\dot{U}=-\frac{1}{R}U+WF(U)+I=G(U,t)$$

为了对此非线性系统进行稳定性分析，方法之一就是在系统的平衡点附近对系统进行线性化处理。设平衡点为 U_e，如果 $G(U,t)$在 U_e 附近连续可微，且存在多阶导数，那么状态矢量中的任意元素 $u_i, i=1,2,\ldots,r$ 满足:

$$\dot{u}_i=g_i(u_e)+\sum_{j=1}^{r}\frac{\partial g_i}{\partial u_j}(u_i-u_{ej})+O_n,\quad i=1,2,\ldots,r$$

若忽略高次微分项 O_n，因 $g_i(u_e)=0$，则上式可写为:

$$\dot{u}_i=\sum_{j=1}^{r}\frac{\partial g_i}{\partial u_j}(u_j-u_{ej}),\quad i=1,2,\ldots,r$$

$\left.\frac{\partial g_i}{\partial u_j}\right|_{u_j=u_{ej}}$ 为雅可比行列式的一个元。由此将非线性方程线性化后，则可用线性方法来分析其稳定性了。

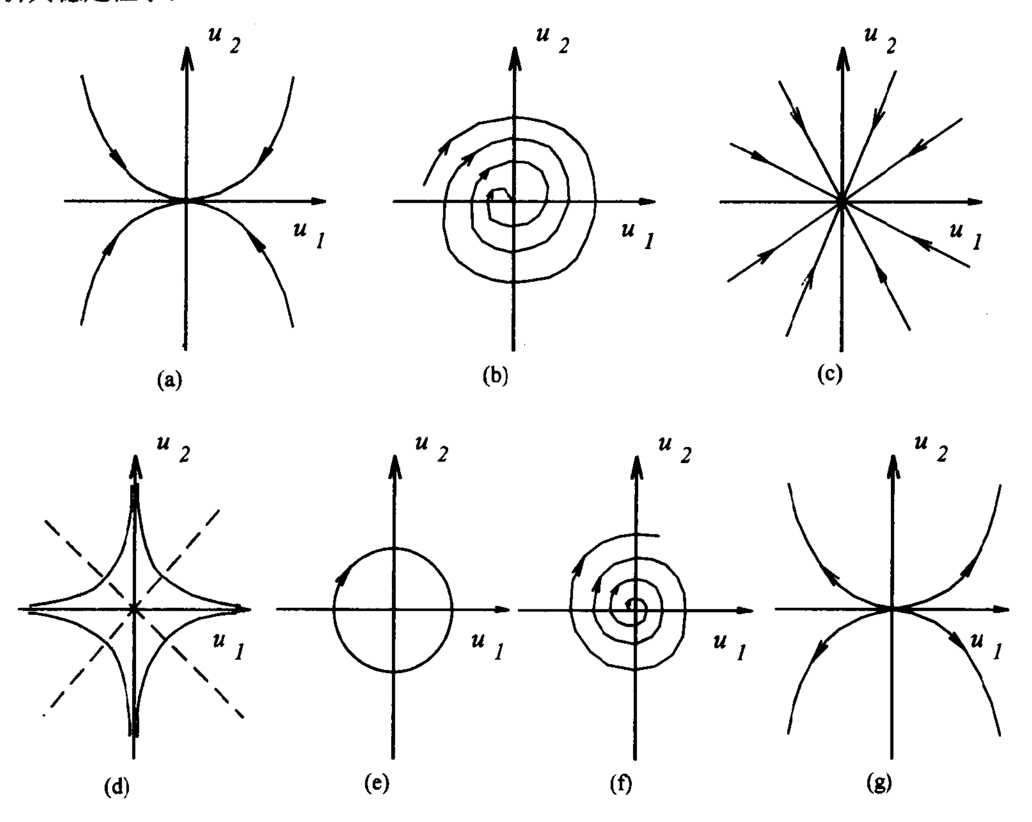


图 5.15　线性反馈网络系统解的情况

下面举一个实际电路的例子来分析其稳定性。

【例 5.8】图 5.16 为由两个运算放大器组成的人工神经网络的互联网电路，其中，

$$v_1=\tanh(\beta\mu_1), v_2=\tanh(\beta\mu_2)$$

分析网络在不同情况下的稳定性。

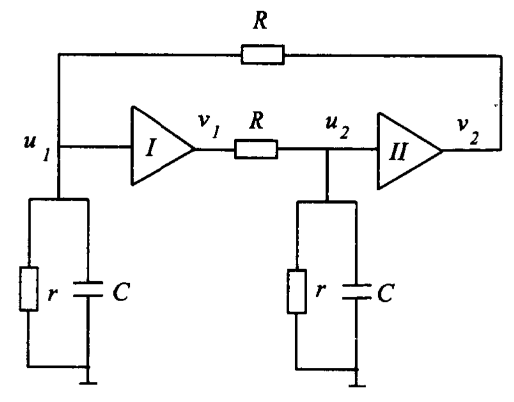


图 5.16　两个神经元的互联电路

解:

由网络结构图，可得其状态方程为:

$$C\frac{\mathrm{d}u_1}{\mathrm{d}t}=\frac{1}{R}(v_2-v_1)-\frac{u_1}{r}$$

$$C\frac{\mathrm{d}u_2}{\mathrm{d}t}=\frac{1}{R}(v_1-v_2)-\frac{u_2}{r}$$

当 β 很小时，可近似为: $v_i=\beta u_i$，β 为放大器增益，因网络完全相同，有 $\beta_1=\beta_2=\beta$。将 $v_i=\beta u_i$ 代入上式，并整理可得:

$$C\begin{bmatrix}\dot{u}_1\\ \dot{u}_2\end{bmatrix}=\frac{1}{R}\begin{bmatrix}-1-\frac{R}{r} & \beta\\ \beta & -1-\frac{R}{r}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

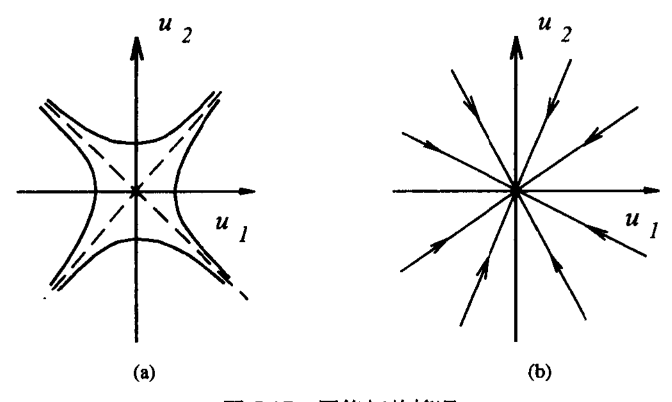
若取 $RC=1$，$r=\infty$，则上式的特征方程为:

$$(\lambda+1)^2-\beta^2=0$$
$$\lambda=\ \beta-1$$

根据β 的不同值，系统状态轨迹可有下列几种情况:

①当 $\beta>1$ 时，有 $\lambda_1>0,\lambda_2<0$，系统的状态轨迹为鞍点。点(0, 0)为系统的一个平衡点。只有当初始值处于 $u_1=-u_2$ 的直线上，才能使系统最终收敛到平衡点上。如图 5.17(a)所示;

②当 $\beta<1$ 时，有 $\lambda_1<0,\lambda_2<0$，此时不论系统的初始状态为何值，均能收敛到唯一的平衡点(0, 0)上。其状态空间上的轨迹如图 5.17(b)所示。



(a) (b)

图 5.17 网络解的情况

从【例 5.8】中可以看出，只有当 $\beta<1$ 时系统才能稳定，否则系统很难稳定。为了能够保证系统的稳定性，无论对什么样的初始条件，都能收敛到系统的稳定点上，我们对【例 5.8】的电路图结构进行改进。

【例 5.9】将【例 5.8】中的第二个运算放大器的输出取反，即有 $v_2=-\beta u_2$，如图 5.18 所示，重新分析系统的稳定性。

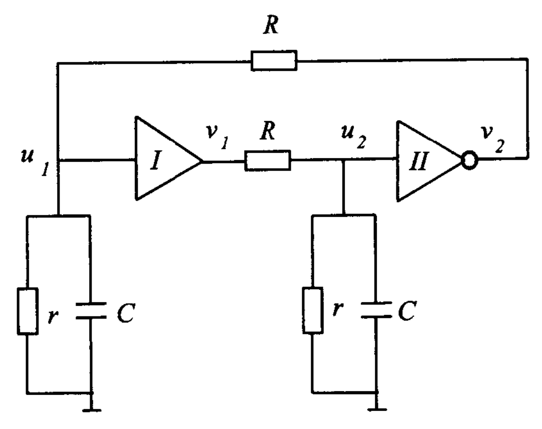


图 5.18　稳定的网络结构图

解:

状态方程得:

$$C\begin{bmatrix}\dot{u}_1\\ \dot{u}_2\end{bmatrix}=\frac{1}{R}\begin{bmatrix}-\frac{1}{R}-\frac{1}{r} & \frac{\beta}{R}\\ \frac{\beta}{R} & -\frac{1}{R}-\frac{1}{r}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

其特征方程为:

$$\lambda^2+2\frac{R+r}{RCr}\lambda+\left(\frac{R+r}{RCr}\right)^2+\frac{\beta^2}{R^2r^2}=0$$

解上述方程得:

$$\lambda_{1,2}=-\frac{R+r}{RCr}\pm\sqrt{\left(\frac{R+r}{RCr}\right)^2-\left(\frac{\beta^2}{R^2r^2}\right)}=-\frac{R+r}{RCr}\pm i\frac{\beta}{RC}$$

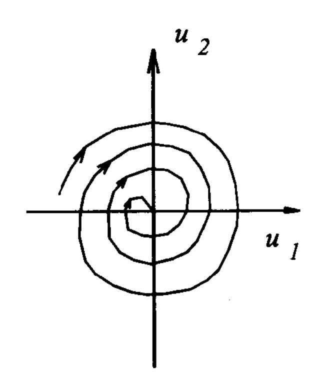


图 5.19　稳定网络状态轨迹图

特征根具有负实部和非零虚部，所以该系统的状态轨迹是一个螺旋递减直至为 0 的曲线。任何初始状态经过系统的运行后，最终趋于稳定的平衡点。

5.4.3 霍普菲尔德能量函数及其稳定性分析

霍普菲尔德在 80 年代初提出了一个对单层反馈动态网络的稳定性判别的函数，这个函数有明确的物理意义，是建立在能量基础上的，同李雅普诺夫函数一样，霍普菲尔德认为在系统的运动过程中，其内部储存的能量随着时间的增加而逐渐减少，当运动到平衡状态时，系统的能量耗尽或变得最小，那么系统自然将在此平衡状态处渐近稳定，即有 $\lim_{t\to\infty} U(t)=U_e$。因此，若能找到一个可以完全描述上述过程的所谓能量函数，那么系统的稳定性问题就可解决。为此对霍普菲尔德反馈网络定义了一种能量函数 E，称为霍普菲尔德能量函数，这个 E 可正可负，但负向有界。所以说，它是李雅普诺夫函数的一种推广，是广义的李雅普诺夫函数。对于连续反馈网络的电路实现其状态方程组为:

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t}=-\frac{u_i}{R}+\sum_{j=1}^{r}w_{ij}v_j+I_i \qquad (5.15)$$
$$v_i=f_i(u_i)$$

当系统达到稳定输出时，霍普菲尔德能量函数定义为:

$$E=-\frac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{r}w_{ij}v_iv_j-\sum_{i=1}^{r}v_iI_i+\sum_{i=1}^{r}\frac{1}{R}\int_0^{v_i}F^{-1}(\eta)\mathrm{d}\eta \qquad (5.16)$$

其中: $\frac{1}{R}=\frac{1}{r}+\sum_{j=1}^{r}w_{ij}$，$w_{ij}$ 为第 j 个输入与第 i 个输入之间的连接导纳，$w_{ij}=\frac{1}{R_{ij}}$；r 与 C 分别为第 i 个运算放大器的电阻和输入电容。 I_i 为外加电流，u_i 和 v_i 分别为第 i 个运算放大器的输入与输出。它们之间的关系为一个单调上升的函数关系，如图 5.20(a)所示，其中β表示运算放大器的放大倍数，图中给出了不同β下输入与输出之间的关系。

在能量函数（5.16）式中，函数 $F^{-1}(v_i)$ 为 u_i 的逆函数。u_i 为 v_i 的函数关系图，如图 5.20(b)所示；能量函数中的积分项表示了输入状态与输出之间关系的能量项，其积分结果如图 5.20(c)所示。

对于理想放大器，（5.16）式所定义的能量函数可以被简化为:

$$E=-\frac{1}{2}\sum_{j=1}^{r}\sum_{j=1}^{r}w_{ij}v_iv_j-\sum_{j=1}^{r}v_iI_i$$

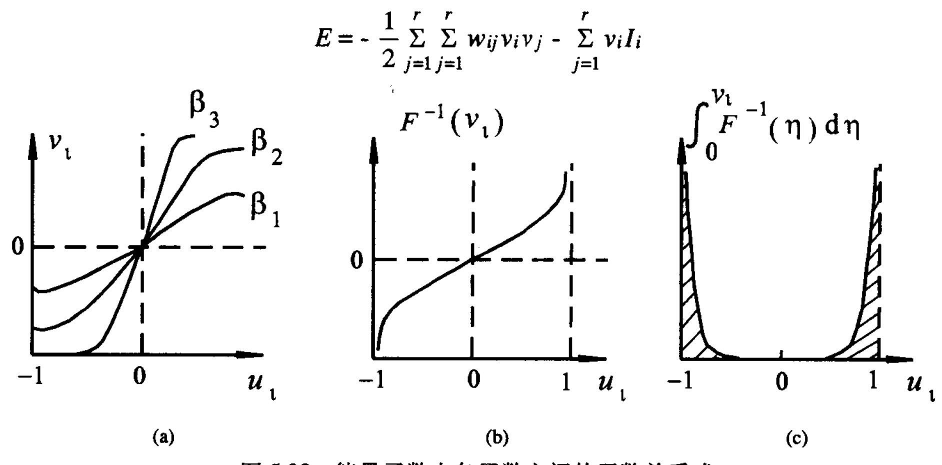


图 5.20 能量函数中各函数之间的函数关系式

这也就成了离散网络的霍普菲尔德能量函数。

对于所定义的霍普菲尔德能量函数(5.16)式有以下结论：对于(5.14)式定义的 CHNN 模型系统，若函数 $v_i = f(u_i)$ 是单调递增且连续可微，则能量函数(5.16)式是单调递减且有界。

下面首先分析一下能量函数的单调递减。

已知：a) $w_{ij} = w_{ji}$；

b) $f(u_i)$ 为单调递减连续函数。

$$
\begin{aligned}
\frac{dE}{dt} &= \sum_{i=1}^{r} \frac{dE}{dv_i} \cdot \frac{dv_i}{dt} \\
&= \sum_i \frac{dv_i}{dt} \sum_i \left[- \sum_i w_{ij} v_j - I_i + \frac{1}{R} f^{-1}(v_1) \right] \\
&= - \sum_i \frac{dv_i}{dt} \sum_i \left[w_{ij} v_j - I_i - \frac{u_i}{R} \right] \\
&= - \sum_i C \cdot \frac{dv_i}{dt} \cdot \frac{du_i}{dt} \\
&= - \sum_i C \cdot \frac{dv_i}{du_i} \cdot \left(\frac{du_i}{dt} \right)^2 \\
&= - \sum_i C \cdot f \cdot (u_i) \cdot \left(\frac{du_i}{dt} \right)^2
\end{aligned}
$$

由已知条件b)可得 $f'(u_i) > 0$，又因为$C > 0$，所以有 $\frac{dE}{dt} < 0$。

当$\frac{dE}{dt}$=0 时，有$\frac{du_i}{dt} = 0$, $i = 1,2,\ldots,r$，这表明，能量E的极小点，与$\frac{du_i}{dt} = 0$的平衡点是一致的。

下面讨论E的有界性。这里最主要的是E不能下降到负无穷大，E在负方向要有界。只有这样，当E随着时间的增加必定会达到一个极限值，而此极限值正是系统的稳定点。

1 ）因为对于状态和输出，有$|v_i| < 1$，w_{ij}是由电子元器件的有界数组成的。所以E中第一项是有界的;

2 ）因为外加电流I_i也是有限值。所以E中第二项是有界的;

3 ）对于E中第三项，式中$F^{-1}(\cdot)$的反映了神经网络的状态与输出之间的关系，也是运算放大器的输入与输出之间的函数关系。若用β 代表运算放大器的增益，那么u_i与v_i的关系可用$v_i = f(u_i, \beta)$来代替$v_i = f(u_i)$，并由此可得 $u_i = \frac{1}{\beta} F^{-1}(v_i)$。

由此可得第三项为$\frac{1}{\beta} \sum_{i=1}^{r} \frac{1}{r} \int_0^{v_i} F^{-1}(\eta) d\eta$，从图 5.18(c)中可以看出，上式的积分对一个i来说在$v_i = 0$时为零，而在其他情况下为正，当$\beta \to \infty$时，$v_i(u_i)$趋向一个符号函数，此时，积分项的作用很小而可以忽略不计，能量函数就由第一、二两项的和决定，成为离散时的情况；E有界。如果β 比较小，第三项为正，它的贡献主要在靠近$v_i \approx \pm 1$的边缘，其值总是小于$2|v_i||u_i|$，所以也是有界的。

当我们对反馈网络应用霍普菲尔德能量函数后，从任意一个初始状态开始，因为在每次迭代后都能满足$\Delta E \le 0$，所以网络的能量将会越来越小，最后趋于稳定点$\Delta E = 0$。霍普

菲尔德能量函数的物理意义是：在那些渐进稳定点的吸引域内，离吸引点越远的状态，所具有的能量越大，由于能量函数的单调下降特性，保证状态的运动方向能从远离吸引点处，不断的趋于吸引点，直到达到稳定点。

几点说明：

①能量函数是反馈网络中一个很重要的概念。根据能量函数，可以很方便的判定系统的稳定性。网络的能量值与其状态存在着一定的联系，即能量的改变对应着状态的变迁，网络的稳定状态对应于能量函数的极小点。正是这种对应关系为网络进行优化计算奠定了基础；

②能量函数与李雅普诺夫函数的区别在于：李氏函数被限定在大于零的范围内，而能量函数无此要求，但要求负向有界；李氏函数要求在零点值为零，即 $v(0)=0$ ，而能量函数无此要求，所以，当能量函数 E 满足 $E(v)<0, E(0)=0,$ 和 $\frac{\mathrm{d}E}{\mathrm{d}t} \leqslant 0$ 时，霍普菲尔德能量函数就是李雅普诺夫函数了；

③霍普菲尔德选择的能量函数，它只是保证系统稳定和渐近稳定的充分条件，而不是必要条件，其能量函数也不是唯一的。为了能够使 $\frac{\mathrm{d}E}{\mathrm{d}t} \leqslant 0$ ，霍普菲尔德对设计权 w_{ij} 有一个对称性的要求，即 $w_{ij}=w_{ji}$ 和 $\frac{\mathrm{d}v_i}{\mathrm{d}u_i}>0$ 的要求。不少文章阐述了即使不满足连接权矩阵对称的条件，仍然可以达到系统的稳定。

5.4.4 能量函数与优化计算

所谓优化问题，是求解满足一定约束条件下的目标函数的极小值问题。有关优化的传统算法很多，如梯度法、单纯型法等，由于在某些情况下，约束条件的过于复杂，加上变量维数较多等诸多原因，使得采用传统算法进行的优化工作耗时过多，有的甚至达不到预期的优化结果。

霍普菲尔德能量函数是一个反映了多维神经元状态的标量函数，而且可以用简单的电路形成人工神经网络，它们的互联形成了并联计算的机制。当各参数设计合理时，由电路组成的系统的状态，可以随时间的变化，最终收敛到渐近稳定点上，并在这些稳定点上使能量函数达到极小值。以此为基础，可以人为地设计出与人工神经网络相对应的电路中的参数，把优化问题中的目标函数、约束条件与霍普菲尔德能量函数联系起来。这样，当电路运行后达到的平衡点，就是能量函数的极小点，其系统状态满足了约束条件下的目标函数的极小值，在此方式下，利用人工神经网络来解决优化问题。由于人工神经网络是并行计算，其计算量不随维数的增加而发生指数性质的“爆炸”，因而特别适用于解决有此问题的优化问题。

5.4.4.1 能量函数设计的一般方法

设优化目标函数为 $f(u), u \in R^T$，为人工神经网络的状态，也是目标函数中的变量。优化的约束条件为： $g(u)=0$ 。优化问题归结为：在满足约束的条件下，使目标函数最小。由于可以设计出等价最小的能量函数 E 为：

$$E=f(u)+\ \Sigma|g_i\ (u)|$$

这里$\Sigma|g_i\ (u)|$也称为惩罚函数，因为在约束条件 $g_i\ (u)=0$ 不能满足时，$\Sigma|g_i\ (u)|$ 的值总是大于零，造成 E 不是最小。

对于目标函数$f(u)$,一般总是取一个期望值与实际值之间之差的平方或绝对值的标量函数，这样能够保证 $f(u)$总是大于零。根据霍普菲尔德能量函数的要求，只有 E 在负的方向上有界，即$|E|<E_{\max}$，同时$\dfrac{dE}{dt}\leqslant 0$，则系统最后总能达到 E 的最小且$\dfrac{dE}{dt}$=0 的点，此点同时又是系统稳定点，即$\dfrac{du_i}{dt}=0$的点。由于求解优化问题的 E 往往是状态 u 的函数，所以，为了求解方便，常常将$\dfrac{dE}{dt}\leqslant 0$的条件转化为对状态求导的条件如下:

$$\frac{\partial E(u_i,v_i)}{\partial u_i}=-\frac{du_i}{dt} \tag{5.17}$$

这是因为，当$\dfrac{du_i}{dt}=-\dfrac{\partial E}{\partial u_i}$，$\dfrac{dE}{dt}=\sum_i\dfrac{dE}{du_i}\ \dfrac{du_i}{dt}=-\sum_i(\dfrac{du_i}{dt})^2\leq 0$

可以说条件（5.17）式是$\dfrac{dE}{dt}\leqslant 0$的另一种表达形式。

同理可得另一个等价的条件为:

$$\frac{\partial E(u_i,v_i)}{\partial v_i}=-\frac{dv_i}{dt}$$

所以在用霍普菲尔德能量函数求解优化问题时，首先应把问题转化为目标函数和约束条件，然后构造出能量函数，并利用条件式求出能量函数中的参数，由此得到人工神经网络的连接权值。

5.4.4.2 具体设计步骤

1）根据要求的目标函数，写出能量函数的第一项$f(u)$;

2）根据约束条件 $g(u)=0$，写出惩罚函数，使其在满足约束条件时为最小，作为能量函数的第二项;

3）加上一项$\sum_i\dfrac{1}{R}\int_0^{v_i}F^{-1}(\eta)d\eta$，此项是人为加上的，因为在神经元状态方程中，存在一项$-\dfrac{u_i}{R}$，它是在人工神经网络的电路实现中产生的。为了使设计出的优化结果能够在电路中得以实现而加上此项。它是一个正值函数，在运行放大增益足够大时，此项可以忽略;

4）根据能量函数 E 求出状态方程，并使下式成立:

$$\frac{\partial E}{\partial u_i}=-\frac{du_i}{dt}$$

5）根据条件与参数之间的关系，求出 w_{ij} 和 b_i, $(i=1,2,...,r)$;

6）求出对应的电路参数，并进行模拟电路的实现。

下面举例说明求解优化问题的模拟人工神经网络的电路实现的过程。

【例 5.10】用人工神经网络来设计一个 4 位 A/D 变换器，要求将一个连续的从 0 到 15

的模拟量 u 变化为输出为 0 或 1 的二进制数字量，即 $v_i \in \{0, 1\}$, v_i 代表第 i 个神经元的输出：$v_i = F(u)$; F 为单调上升有限量函数。

解:

A/D 转换器的实质是对于给定的模拟量输入，寻找一个二进制数字量输出，使输出值与输入模拟量之间的差为最小。传统的转换方式，只要对模拟输入量用 2 碾转相除，记录余数，就可得到二进制的变换值。采用人工神经网络进行转换，首先需定义能量函数。一个四位 A/D 转换器可以用具有四个输出节点的 CHNN 来实现。

假定神经元的输出电压 v_i $(i = 1,2,3)$可在 0 与 1 之间连续变化，当网络达到稳态时，各节点的输出为 0 或 1，若此时的输出状态所表示的二进制值与模拟输入量相等，则表明此人工神经网络达到了 A/D 变换器的功能。输入与输出之间的关系则满足下式:

$$\sum_{i=0}^{3} v_i 2^i = u$$

由此可见，要使 A/D 转换器结果 $v_3v_2v_1v_0$ 为输入 u 的最佳数字表示，必须满足两点:

①每个输入 v_i 必须趋于 0 值或 1 值，至少比较接近这两个数;

② $v_3v_2v_1v_0$ 的和值应尽量接近值 u。

为此，利用最小方差的概念，对输出 v_i 按下列指标来选取 $f(u)$:

$$f(u) = \frac{1}{2}(u - \sum_{i=0}^{3} v_i 2^i)^2 > 0$$

上式中，u 为输入的模拟量，v_i 为数字量，对于 $i = 0,1,2,3$, $v_i \in \{0, 1\}$; 2^i 表示二进制数的位数，此目标函数大于零，所以 $f(u)$存在极小值，且当 $f(u) = f_{\min}(u)$时，v_i 为 u 的正确转换。

将 $f(u)$展开，并整理后得:

$$f(u) = \frac{1}{2}\sum_{i=0}^{3}\sum_{j=0}^{3}(2^{i+j} v_i v_j) - \sum_{i=0}^{3} 2^i u v_i + \frac{1}{2}u^2$$

仅用一项 $f(u)$并不能保证 v_i 的值充分接近逻辑值 0 或 1，因为可能存在其他 v_i 值,(v_i 可以在[0, 1]中连续变化）使 $f(u)$为最小，为此，增加一个约束条件:

$$g(u) = -\frac{1}{2}\sum_{i=0}^{3}(2^i)^2(v_i - 1)v_i$$

$g(u)$保证了输出只有在{0, 1}时取最小值零，而对于 v_i 为 0 与 1 之间的实数时，$g(u) = 0$。所以此约束条件保证了输出的数字量 0 或 1。

a)写出能量函数 E

$$E = f(u) + g(u) + \sum_{i=0}^{3}\frac{1}{R_i}\int_0^{V_i} F^{-1}(\eta)\mathrm{d}\eta$$

$$= \frac{1}{2}(u - \sum_{i=0}^{3} v_i 2^i)^2 - \frac{1}{2}\sum_{i=0}^{3}(2^i)^2(v_i - 1)v_i + \sum_{i=0}^{3}\frac{1}{R_i}\int_0^{v_i} F^{-1}(\eta)\mathrm{d}\eta$$

如前所述，考虑到电路的具体实现在 E 上加了一个大于零的分项。十分明显，E 是大于零的即有下界。

b) 计算$\dfrac{\mathrm{d}u_i}{\mathrm{d}t}$

因为:

$$\frac{\mathrm{d}v_i}{\mathrm{d}t} = F'\ (u_i)\frac{\mathrm{d}u_i}{\mathrm{d}t}$$

而$F'\ (U_i) > 0$。在放大区内可以近似为一个常数C，即：$F'\ (U_i)$=C。

所以有:

$$\frac{\partial E}{\partial v_i} = -\frac{\mathrm{d}v_i}{\mathrm{d}t} = -C\frac{\mathrm{d}u_i}{\mathrm{d}t}$$

而

$$\frac{\partial E}{\partial v_i} = \sum_{\substack{j=0\\ j\ \ i}}^{3} 2^{i+j} v_j - 2^i u + 2^{2i-1} + \frac{u_i}{R} = -C\frac{\mathrm{d}u_i}{\mathrm{d}t}$$

重写上式得:

$$C\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{u_i}{R} + \sum_{\substack{j=0\\ j\ \ i}}^{3} (-2^{i+j})v_j + (-2^{2i-1} + 2^i u)$$

c) 将上式与实现电路的状态方程组相比较可得:

$$w_{ij} = \begin{cases} -2^{i+j}, & i \neq j \\ 0\ , & i = j \end{cases} \quad ; \quad I_i = -2^{2i-1} + 2^i u$$

即

$$w_{01} = w_{10} = -2; \quad w_{02} = w_{20} = -4; \quad w_{03} = w_{30} = -8; \quad w_{12} = w_{21} = -8;$$

$$w_{13} = w_{31} = -16; \quad w_{23} = w_{32} = -32; \quad w_{00} = w_{11} = w_{22} = w_{33} = 0.$$

$$I_0 = -0.5 + u;$$

$$I_1 = -2 + 2u;$$

$$I_2 = -8 + 4u;$$

$$I_3 = -32 + 8u.$$

根据上面的设计，可以设计出完成优化求解的模拟电路人工神经网络系统，如图 5.21 所示。权的负值是通过倒相运算放大器来完成的，所得到的负输出再用一次倒相变正，图中标出的数据是以导纳表示 w_{ij} 的值。

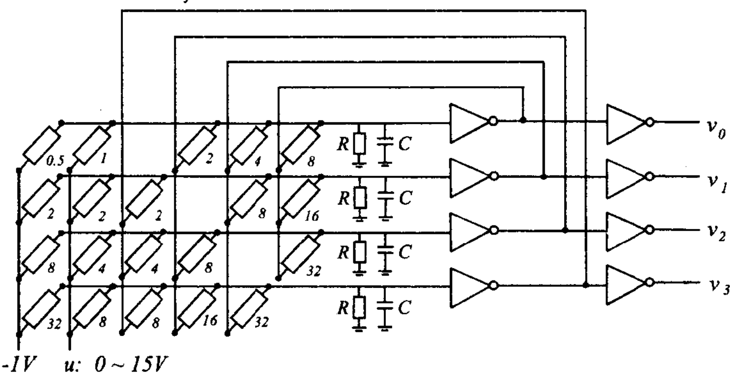


图 5.21 固定导纳矩阵的 4 位 A/D 变换实现电路

在每次运行网络之前，应使状态复位为零，这样，当给网络输入 u 值时，网络的运行后的稳定输出即为 u 的正确二进制输出。若运行前没有进行复位，那么在下一次进行转换时，状态仍停留在上次转换的输出状态，即局部极小值上，因而难以跳出，可能造成错误的转换。

当输入 u 为 0 ~ 15V 数字时，网络对应可转换为 0000 ~ 1111 之间的数，这种输入模拟量与输入数字量之间具有 5.22 图的关系。

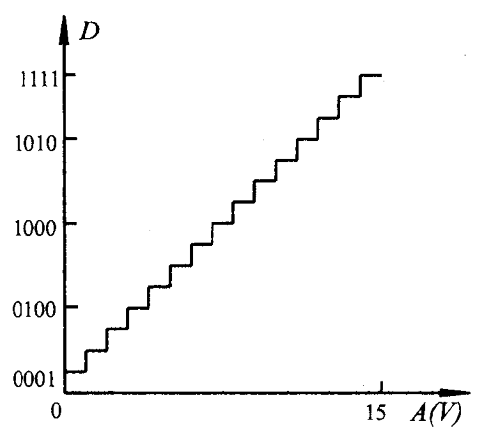


图 5.22　A/D 转换输入/输出关系图

从这个例子可以看出，优化问题的求解是设法将问题转化为能量函数的构造，一旦对应的能量函数构造成功，人工神经网络电路也就设计出来，最优解就可以随之解出来了。但这种能量函数的构造没有现成的公式，而是一种技能，凭经验而熟能生巧，为此下面再举一个利用能量函数求解优化问题的例子。

【例 5.11】 TSP 问题。

所谓 TSP （ Travelling Saleman Problem ）问题，即“旅行商问题”是一个十分有名的难以求解的优化问题，其要求很简单：在 n 个城市的集合中，找出一条经过每个城市各一次，最终回到起点的最短路径。

如果已知城市 A， B， C， D，...，之间的距离为 $d_{AB}, d_{BC}, d_{CD}, \ldots$，那么总的距离 $d = d_{AB} + d_{BC} + d_{CD} + \ldots$，对于这种动态规化问题，要去求其 $\min(d)$的解。因为对于 n 个城市的全排列共有 n 种，而 TSP 并没有限定路径的方向，即为全组合，所以对于固定的城市数 n 的条件下，其路径总数 S_n 为 $S_n = n! / 2n\ (n \geqslant 4)$，例如 $n = 4$ 时， $S_n = 3$，即有三种方式，如图 5.23 所示。

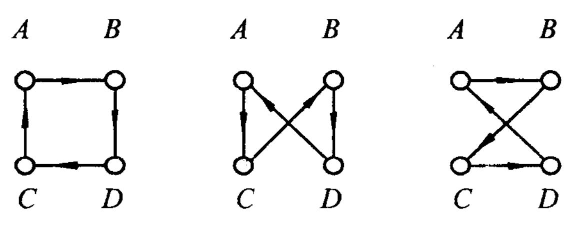


图 5.23　$n = 4$ 时的 TSP 路径图

表 5.2 城市数和对应的旅行方案数

城市数	旅行方案数 = $n!/2n$
3	1
4	3
5	12
6	60
7	360
8	2520
9	20160
10	181440
11	1814400
12	19958400
13	239500800
14	3113510400
15	4.3589145×10^{10}
16	6.5383718×10^{11}
17	1.0461394×10^{13}
18	1.7784371×10^{14}

由斯特林(Stirlin)公式，路径总数可写为:

$$S_n = \frac{1}{2n}[\sqrt{2\pi n} \cdot e^{n(\ln n-1)}]$$

若采用穷举搜索法，则需要考虑所有可能的情况，找出所有的路径，再分别对其进行比较，以找出最佳路径，因其计算复杂程度随城市数目的增加呈指数增长，可能达到无法进行的地步。从表 5.2 中可以看到，当城市数为 16 时，旅行方案数已超过 6×10^{11} 种。而每增加一个城市，所增加的方案数为:

$$\frac{(n+1)!}{2(n+1)} \Big/ \frac{n!}{2n} = n$$

这类问题称为完全非确定性多项式问题（ Nondeterministic Polynomial Complete, 简称 NP 完全问题）。由于求解最优解的负担太重，通常比较现实的作法是求其次优解。霍普菲尔德网络正是一种合适的方法。因为它可以保证其解向能量函数的最小值方向收敛，但不能确保达到全局最小值点。

采用连续时间的霍普菲尔德网络模型来求解 TSP ，开辟了一条解决这一问题的新途径。其基本思想是把 TSP 映射到 CHNN 上，通过网络状态的动态演化逐步趋向稳态而自动地搜索出优化解。为了便于神经模型的实现，必须首先找到过程的一个合适的表达方法。TSP 的解是若干城市的有序排列，任何一个城市在最终路径上的位置可用一个 n 维的 0 、 1 矢量表示，对于所有 n 个城市，则需要一个 $n \times n$ 维矩阵，例如以 5 个城市为例，一种可能的排列矩阵为:

7	1	2	3	4	5
A	0	1	0	0	0
B	0	0	0	1	0
C	1	0	0	0	0
D	0	0	1	0	0
E	0	0	0	0	1

其中，行矢量表示城市名，列矢量表示城市中在旅行中排的序号。该矩阵唯一地确定了一条有效的行程路径：

$$C \to A \to D \to B \to E$$

很明显，为了满足约束条件，该矩阵中每一行以及每一列中只能有一个元素为 1 ，其余元素均为零。这个矩阵称为关联阵。若用 d_{xy} 表示从城市 x 到城市 y 的距离，则上面路径的总长度为：

$$d_{xy} = d_{CE} = d_{AD} = d_{DB} = d_{BE}$$

TSP 的最优解是求长度 d_{xy} 为最短的一条有效的路径。为了解决 TSP，必须构造这样一个网络：在网络运行时，其能量能不断降低。在运行稳定后，网络输出能代表城市被访问的次序，即构成上述的关联矩阵。网络能量的最小值对应于最佳（或次最佳）的路径距离。所以解决问题的关键，仍然是构造合适的能量函数。

（1）目标函数 $f(V)$

对于一个 n 城市的 TSP，需要 $n \times n$ 节点的 CHNN 。假设每个神经元的输出记为 V_{xi}，V_{yi}，行下标 x 和 y 表示不同的城市名，列下标 i 和 j 表示城市在路径中所处的次序位置，通过 V_{xi}，V_{yi} 取 0 或 1，可以通过关联矩阵确定出不同种的访问路径。用 d_{xy} 表示两个不同的城市之间的距离，对于选定的任一 V_{xi}，和它相邻的另一个城市 y 的状态可以有 $V_{y(i+1)}$ 和 $V_{y(i-1)}$。那么，目标函数 $f(V)$ 可选为：

$$f(V) = \frac{P}{2} \sum_{x} \sum_{\substack{y \\ x \neq y}} \sum_{i} d_{xy} V_{xi} (V_{y(i+1)} + V_{y(i-1)})$$

这里所选择的 $f(V)$ 表示的是对应于所经过的所有路径长度的总量，其数值为一次有效路径总长度的倍数，当路径为最佳时，$f(V)$ 达到最小值，它是输出的函数。

当 $V_{xi} = 0$ 时，则有：$f(V) = 0$，此输出对 $f(V)$ 没有贡献；当 $V_{xi} = 1$ 时，则通过与 i 相邻位置的城市 $i+1$ 和 i -1 的距离，如在关联矩阵中 $V_{D3} = 1$，那么，与 $i = 3$ 相邻位上的两个城市分别为 V_{A2} 和 V_{B4}，此时在 $f(V)$ 中可得到 d_{AD} 和 d_{DB} 两个相加的量，依此类推，把推销员走过的全部距离全加起来，即得 $f(V)$。

（2）约束条件 $g(V)$

约束条件要保证关联矩阵的每一行每一列中只有一个值为 1 ，其他值均为零，用三项表示为：

$$g(V) = \frac{Q}{2} \sum_{x} \sum_{\substack{y \\ x \neq y}} \sum_{i} V_{xi} V_{yi} + \frac{S}{2} \sum_{x} \sum_{y} \sum_{\substack{i \\ i \neq j}} d_{xi} V_{xj} + \frac{T}{2} \sum_{x} \sum_{y} (V_{xi} - n)^{2}$$

其中：

第一项，当且仅当关联矩阵每一列包含不多于一个 1 元素时，此项为最小；

第二项，当且仅当关联矩阵每一行包含不多于一个 1 元素时，此项为最小；

第三项，当且仅当关联矩阵中元素为 1 的个数为 n 时，此项为最小。

即 $g(V)$ 保证满足了所有三项要求，收敛到有效解时其值为 0 。

（3）总的能量函数 E

$$E = f(V) + g(V)$$

选择使用高增益放大器，这样能量函数中的积分分项可以忽略不计，求解得网络的联接权值为：

$$\begin{aligned} w_{xi,yi} = & - S\delta_{xy}(1-\delta_{ij}) && \text{行抑制} \\ & - Q\delta_{ij}(1-\delta_{xy}) && \text{列抑制} \\ & - T && \text{全局抑制} \\ & - Pd_{xy}(\delta_{j(i+1)}+\delta_{j(i-1)}) && \text{路径长度} \end{aligned}$$

式中：

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

外部输入偏置电流为：

$$I_{xi} = C$$

求解 TSP 的连接神经网络模型的运动方程可表示为：

$$\begin{cases} \mathrm{d}U_{xi} = -S\sum_{j\neq i}V_{xy} - Q\sum_{j\neq x}V_{yi} - T\sum_{x}\sum_{j}(V_{xj}-n) \\ \qquad - P\sum_{y}d_{xy}(V_{y(i+1)}+V_{y(i-1)}) - \dfrac{U_{xi}}{R_{xi}C_{xi}} \\ V_{xi} = f(U_{xi}) = \dfrac{1}{2}[1+th(\dfrac{U_{xi}}{U_0})] \end{cases}$$

式中 U_0 为初始值，非性线函数取近似于 S 型的双曲正切函数。霍普菲尔德和泰克(Tank)经过实验，认为取初始值为：$S = Q = P = 500, T = 200, RC = 1, U_0 = 0.02$ 时，其求解 10 个城市的 TSP 得到良好的效果。人们后来发现，用连续霍普菲尔德网络求解像 TSP 这样约束优化问题时，系统 S、Q、P、T 的取值对求解过程有很大影响。

5.5 本章小结

对于联想记忆问题，动态网络比前向网络有更强的适应性。一般来讲，联想记忆中不仅要求把需存储的输入样本存储起来，并能予以恢复，而且要求网络对所存样本中的每一个样本有足够大的吸引域，以便对模糊、畸变、缺损信息，能予以复原和补全。在反馈网络的设计中，其系统分析是网络设计的基础。系统分析工作可使人们能对网络模型有更清楚更全面的认识。它包括特性分析和功能分析两个方面。特性分析包括：拓扑结构、网络容量、算法分析、求解稳定性、收敛性、计算复杂性分析等；功能分析包括：联想记忆能力、自组织、自学习和自适应能力、处理模糊、随机、缺损信息能力、识别各类模式的能力等。

对离散的反馈网络，若 $w_{ij} = w_{ji}$（系统对称联接），则在网络演化过程中能量 E 必然单调下降。由于 E 有界，网络必然趋于某个稳态，能量 E 达局域极小值对应于系统的某个定点吸引子。若将定点吸引子作为所存样本，则能量 E 的各极值点代表一系列存储内容，网络的初态（包括各不稳定状态点）可视作某记忆事物的已知部分，能量函数的变化则对应于存储内容的联想过程。网络状态从非稳态点向稳态点的演化过程相当于由事物的部分

信息自动联想出事物的全部整体的过程。离网络初态最近的那个吸引子是网络所复原的某存储内容，一般这类网络具有多个定点吸引子，各自具有一定的吸引域。系统能量空间好似一个坑坑洼洼的大盆，坑底为定点吸引子，坑的大小为吸引域范围，坑越多，定点吸引子也越多。选定网络初态恰似丢一个小球至某坑的边缘，小球落在哪个坑边缘就掉向那个坑底。坑越大，即吸引域越大，越容易接纳小球至坑底，故联想范围就越广。因此，联想存储器的定点吸引子越多，吸引域越大，可存储的内容就越多，联想范围（即抗噪声畸变能力）也越大。

若网络的联接矩阵中各元素 w_{ij} 不对称，那么情况就比较复杂，除了定点吸引子外，还会有极限环吸引子，甚至会有小范围的混沌吸引子。因此，需对此作系统的进一步分析。对于给定的网络，系统分析所涉及的工作是研究相空间随时间增长而不断收缩的过程，终态集的大小（吸引子数目）、吸引子类型、吸引域大小及其形状等。

令 A_e 代表定点吸引子，A_m 代表 m 周期吸引子，有下述结论成立：

1）如权矩阵对称，且正定，则反馈网络同步运行收敛，只有 A_e 解；

2）如权矩阵对称，且为零对角阵，则网络异步运行收敛，只有 A_e 解；

3）如权矩阵对称，且负定，则网络同步运行只有 A_m 解；

4）如权矩阵对称，则网络同步运行只可能有 A_e 或 A_m 解，或 A_e 和 A_m 解同时存在。

上述这些结论是动态网络系统分析的基础。由于人工神经网络系统理论尚不完善，还不可能对所有动态网络的动态学性质给出结论，故在实际的系统分析工作中还存在较大的理论困难，急待人们去探索。事实上，根据上述结论设计的网络，适用范围很小。而且上述判据仅是充分条件而非必要条件。除了定点吸引子（对应网络收敛性）性质，即存储状态点的稳定性外，联想存储网络的存储容量、伪吸引子以及存储定点吸引子的吸引域形状也是系统分析的重要内容。网络存储容量与定点吸引子数目有关，但单纯讨论存储容量是没有意义的，还需考虑吸引域的大小和形态以及伪吸引子的数量，只有这样才能对网络模型作全面的评估。而网络存储容量的增加，存储吸引子的吸引域很快缩小，而无存储内容的伪吸引子数量及其吸引域范围会很快地增加。当网络存储容量增大到一定程度时，相空间绝大部分区域成为伪吸引子区域。此时，存储样本的网络稳态点吸引域极小，在某种意义上已失去联想功能。另外，在网络存储容量增加的情况下，不但存储态吸引子的吸引域缩小，而且其吸引域形状变得复杂，各向同性程度明显变差，故联想能力下降。通过上述系统分析可见，霍普菲尔德网络用作联想存储时性能较差，网络容量小，不适宜做大规模内容的联想记忆存储器。

习 题

〖5.1〗试设计一个反馈网络存储下列目标平衡点：

$$T = [1\ -1; -1\quad 1];$$

并用6组任意随机初始列矢量作为输入矢量对所设计的网络的平衡点进行测试，观察3次循环的每一次的输出结果。

第六章　自组织竞争人工神经网络

在实际的神经网络中，比如人的视网膜中，存在着一种“侧抑制”现象，即一个神经细胞兴奋后，通过它的分支会对周围其他神经细胞产生抑制．这种侧抑制使神经细胞之间出现竞争，虽然开始阶段各个神经细胞都处于程度不同的兴奋状态，由于侧抑制的作用，各细胞之间相互竞争的最终结果是：兴奋作用最强的神经细胞所产生的抑制作用战胜了它周围所有其他细胞的抑制作用而“赢”了，其周围的其他神经细胞则全“输”了。

自组织竞争人工神经网络正是基于上述生物结构和现象形成的。它能够对输人模式进行自组织训练和判断，并将其最终分为不同的类型。与 BP 网络相比，这种自组织自适应的学习能力进一步拓宽了人工神经网络在模式识别、分类方面的应用，另一方面，竞争学习网络的核心——竞争层，又是许多种其他神经网络模型的重要组成部分，例如科荷伦(Kohonen)网络（又称特性图）、反传网络以及自适应共振理论网络等中均包含竞争层。

6.1　几种联想学习规则

格劳斯贝格（ S. Grossberg ）提出了两种类型的神经元模型：内星与外星，用以来解释人类及动物的学习现象。一个内星可以被训练来识别一个矢量；而外星可以被训练来产生矢量。

由 r 个输人构成的格劳斯贝格内星模型如图 6.1 所示。

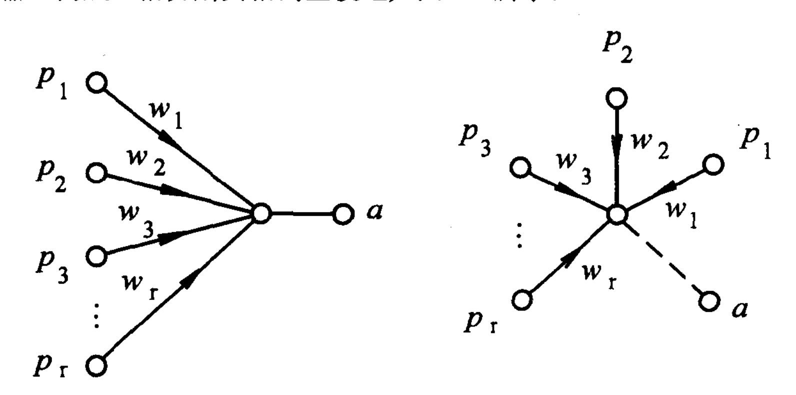


图 6.1　格劳斯贝格内星模型图

由 s 个输出节点构成的格劳斯贝格外星模型如图 6.2 所示。

从图 6.1 和图 6.2 中可以清楚地看出，内星是通过联接权矢量 W 接受一组输人信号 P；而外星则是通过联接权矢量向外输出一组信号 A 。它们之所以被称为内星和外星，主要是因为其网络的结构像星形，且内星的信号流向星的内部；而外星的信号流向星的外部。下

面分别详细讨论两种神经元模型的学习规则及其功效。

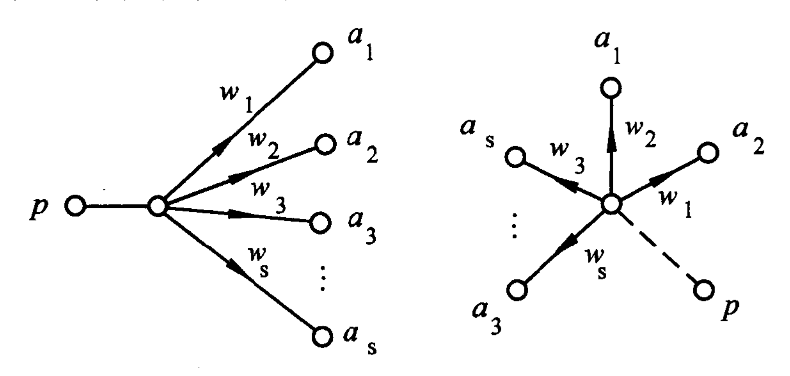


图 6.2　格劳斯贝格外星模型图

6.1.1　内星学习规则

实现内星输入/输出转换的激活函数是硬限制函数。可以通过内星及其学习规则来训练某一神经元节点只响应特定的输入矢量 P，它是借助于调节网络权矢量 W 近似于输入矢量 P 来实现的。

在图 6.1 所示的单内星中对权值修正的格劳斯贝格内星学习规则为：

$$\Delta w_{1j} = lr \cdot (p_j - w_{1j}) \cdot a\text{，}\quad j = 1, 2, \ldots, r \tag{6.1}$$

由（6.1）式可见，内星神经元联接强度的变化。Δw_{1j} 是与输出成正比的。如果内星输出 a 被某一外部方式而维护高值时，那么通过不断反复地学习，权值将能够逐渐趋近于输入矢量 p_j 的值，并趋使 Δw_{1j} 逐渐减少，直至最终达到 $w_{1j} = p_j$，从而使内星权矢量学习了输入矢量 P，达到了用内星来识别一个矢量的目的。另一方面，如果内星输出保持为低值时，网络权矢量被学习的可能性较小，甚至不能被学习。

现在来考虑当不同的输入矢量 P^1 和 P^2 分别出现在同一内星时的情况。首先，为了训练的需要，必须将每一输入矢量都进行单位归一化处理，即对每一个输入矢量 P^q（$q = 1, 2$），用 $1\Big/\sqrt{\sum_{j=1}^{r}(p_j^q)^2}$ 去乘以每一个输入元素，因此所得的用来进行网络训练的新输入矢量具有单位 1 的模值。

当第一个矢量 P^1 输入给内星后，网络经过训练，最终达到 $W = (P^1)^T$。此后，给内星输入另一个输入矢量 P^2，此时内星的加权输入和为新矢量 P^2 与已学习过矢量 P^1 的点积，即

$$N = W \cdot P^2 = (P^1)^T \cdot P^2 = \| P^1 \| \| P^2 \| \cos\theta_{12} = \cos\theta_{12} \tag{6.2}$$

因为输入矢量的模已被单位化为 1，所以内星的加权输入和等于输入矢量 P^1 和 P^2 之间夹角的余弦。

根据不同的情况，内星的加权输入和可分为如下几种情况：

1）P^2 等于 P^1，即有 $\theta_{12} = 0$，此时，内星加权输入和为1；

2）P^2 不等于 P^1，随着 P^2 向着 P^1 离开方向的移动，内星加权输入和将逐渐减少，直到 P^2

与P^1垂直，即θ_{12}=90° 时，内星加权输入和为0;

3）当$P^2 = -P^1$，即θ_{12}= 180° 时，内星加权输入和达到最小值-1。

由此可见，对于一个已训练过的内星网络，当输入端再次出现该学习过的输入矢量时，内星产生 1 的加权输入和；而与学习过的矢量不相同的输入出现时，所产生的加权输入和总是小于 1 。如果将内星的加权输入和送入到一个具有略大于-1 偏差的二值型激活函数时，对于一个已学习过或接近于已学习过的矢量输入时，同样能够使内星的输出为 1 ，而其他情况下的输出均为 0 。所以在求内星加权输入和公式中的权值 W 与输入矢量 P 的点积，反映了输入矢量与网络权矢量之间的相似度，当相似度接近 1 时，表明输入矢量 P 与权矢量相似，并通过进一步学习，能够使权矢量对其输入矢量具有更大的相似度，当多个相似输入矢量输入内星，最终的训练结果是使网络的权矢量趋向于相似输入矢量的平均值。

内星网络中的相似度是由偏差b来控制，由设计者在训练前选定，典型的相似度值为b = -0.95，这意味着输入矢量与权矢量之间的夹角小于18° 48′ 。若选b = -0.9时，则其夹角扩大为25° 48′ 。

一层具有 s 个神经元的内星，可以用相似的方式进行训练，权值修正公式为:

$$\Delta w_{ij} = lr \cdot (p_j - w_{ij}) \cdot a \tag{6.3}$$

MATLAB 神经网络工具箱中内星学习规则的执行是用函数 **learnis.m** 来完成上述权矢量的修正过程:

```
dW = learnis ( W, P, A, lr );
W = W + dW;
```

一层 s 个内星神经元可以作为一个 r 到 1 的解码器。另外，内星通常被嵌在具有外星或其他元件构成的大规模网络中以达到某种特殊的目的。

下面给出有关训练内星网络例子。

【例 6.1 】设计内星网络进行以下矢量的分类辨识:

```
P = [ 0.1826   0.6325;
      0.3651   0.3162;
      0.5477   0.3162;
      0.7303   0.6325 ];
T = [ 1 0 ];
```

与感知器分类功能不同，内星是根据期望输出值，在本例题中是通过迫使网络在第一个输入矢量出现时，输出为 1 ，同时迫使网络在第二个输入矢量出现时，输出为 0 ，而使网络的权矢量逼近期望输出为 1 的第一个输入矢量。

我们首先对网络进行初始化处理:

```
[ R, Q ] = size ( P );
```

```
[ S, Q ] = size ( T );
W = zeros ( S, R );
B = - 0.95 * ones ( S, 1 );
max_epoch = 10;
lr = 0.7;
```

注意权矢量在此是进行了零初始化，这里的学习速率的选择也具有任意性，当输入矢量较少时，学习速率可以选择较大以加快学习收敛速度。另外，因为所给例题中所给输入矢量已是归一化后的值，所以不用再作处理。下面是设计训练内星网络的程序:

```
for epoch = 1 : max_epoch
    for q = 1 : Q
    A = T ( :, q );
    dW = learnis ( W, P( :, q ), A, lr );
    W = W + dW;
    end
end
```

经过 10 次循环以及 480 次计算后，得到的权矢量为:

```
W = [ 0.1826   0.1651   0.5477   0.7303  ;
        0         0        0         0   ]
```

而当 lr = 0.3 时，其结果为:

```
W = [ 0.1805   0.3609   0.5414   0.7219  ;
        0         0        0         0   ]
```

由此可见，学习速率较低时，在相同循环次数下，其学习精度较低。但当输入矢量较多时，较大的学习速度可能产生波动，所以要根据具体情况来确定参数值。

在此例题中，因为只有一个输入矢量被置为 1，所以实际上，所设置的偏差 $B = -0.95$ 没有起到作用。内星网络常用在竞争网络以及后面所要介绍的自适应共振理论 ART 网络中。在那里，其网络的期望输出是通过竞争网络的竞争而得到的。在 ART 网络中，竞争获胜节点的有效性是通过所给的相似度 B 值来决定的。

6.1.2 外星学习规则

外星网络的激活函数是线性函数，它被用来学习回忆一个矢量，其网络输入 P 也可以是另一个神经元模型的输出。外星被训练来在一层 s 个线性神经元的输出端产生一个特别

的矢量 A 。所采用的方法与内星识别矢量时的方法极其相似。

对于一个外星，其学习规则为:

$$\Delta w_{i1} = lr \cdot (a_i - w_{i1}) \cdot p_j \tag{6.4}$$

与内星不同，外星联接强度的变化 ΔW 是与输入矢量 P 成正比的。这意味着当输入矢量被保持高值，比如接近 1 时，每个权值 w_{i1} 将趋于输出 a_i 值，若 $p_j = 1$，则外星使权值产生输出矢量。

当输入矢量 p_j 为 0 时，网络权值得不到任何学习与修正。

当有 r 个外星相并联，每个外星与 s 个线性神经元相连组成一层外星时，每当某个外星的输入节点被置为 1 时，与其相连的权值到矢量 w_{ij} 就会被训练成对应的线性神经元的输出矢量 A，其权值修正方式为:

$$\Delta W = lr \cdot (A - W) \cdot P \tag{6.5}$$

其中:

$W = s \times r$ 权值列矢量;

$lr =$ 学习速率;

$A = s \times q$ 外星输出;

$P = r \times q$ 外星输入。

MATLAB 工具箱中实现外星学习与设计的函数为 **learnos.m**,其调用过程如下:

```
dW = learnos ( W, A, P, lr );
W = W + dW;
```

下面给出外星的一个例题。

【例 6.2 】下面有两元素的输入矢量以及与它们相关的四元素目标矢量，试设计一个外星网络实现有效的矢量的获得，外星没有偏差。

```
P = [ 1   0 ];
T = [ 0.1826   0.6325;
      0.3651   0.3162;
      0.5477   0.3162;
      0.7303   0.6325 ];
```

很显然，此例题为内星【例 6.1 】的反定义。

该网络的每个目标矢量强迫为网络的输出，而输入只有 0 或 1 。网络训练的结果是使其权矩阵趋于所对应的输入为 1 时的目标矢量。

同样网络被零权值初始化:

```
[ R, Q ] = size ( P );
```

```
[ S, Q ] = size ( T );
W = zeros ( S, R );
max_epoch = 10;
lr = 0.3;
```

下面根据外星学习规则进行训练:

```
for epoch = 1 : max_epoch
    for q = 1 : Q
    A = T ( :, q );
    dW = learnos ( W, P( :, q ), A, lr );
    W = W + dW;
    end
end
```

一旦训练完成，当外星工作时，对设置于输入为 1 的矢量，将能够回忆起被记忆在网络中的第一个目标矢量的近似值:

```
》 Ptest = [ 1 ];
》 A = purelin (W*Ptest)
A =
    0.1774
    0.3548
    0.5322
    0.7097
```

由此可见，此外星已被学习来回忆起了第一个矢量。事实上，它被学习来回忆出在【例 6.1 】中学习识别出的那个矢量。即上述外星的权值非常接近于【例 6.1 】中已被识别的矢量。

内星与外星之间的对称性是非常有用的。对一组输入和目标来训练一个内星层与将其输入与目标相对换，来训练一个外星层的结果是相同的，即它们之间的权矩阵的结果是相互转置的。这一事实，被后来应用到了 ART1 网络中。

6.1.3 科荷伦学习规则

科荷伦学习规则是由内星规则发展而来的。对于其值为 0 或 1 的内星输出，当只对输出为 1 的内星权矩阵进行修正，即学习规则只应用于输出为 1 的内星上，将内星学习规则中的 a_i 取值 1 ，则可导出科荷伦规则为:

$$\Delta w_{ij} = lr \cdot (p_j - w_{ij}) \tag{6.6}$$

科荷伦学习规则实际上是内星学习规则的一个特例，但它比采用内星规则进行网络设计要节省更多的学习，因而常常用来替代内星学习规则。

在 MATLAB 工具箱中，在调用科荷伦学习规则函数 **learnk.m** 时，一般通过先寻找输出为 1 的行矢量 i，然后仅对与 i 相连的权矩阵进行修正。使用方法如下:

```
i = find ( A == 1);
dW = learnk ( W, P, i, lr );
W = W + dW;
```

【例 6.3】用科荷伦学习规则重新训练【例 6.1】，观察训练循环的次数。

【例 6.1】中采用内星学习规则使网络权矢量识别了输人矢量，共花费了 10 次循环，共 480 次运算，这并不算多。不过当采用科荷伦学习规则对其进行设计时，其计算量需要得更少。

采用设计训练内星和外星类似的方式，可以非常简单迅速地写出主要的训练程序:

```
for epoch = 1 : max_epoch
    for q = 1 : Q
    A = T ( :, q );
    i = find ( A == 1);
    dW = learnk ( W, P ( :, q ), i, lr );
    W = W + dW;
    end
end
```

这次，在 10 次循环中只用了 280 次运算就得到了相同的结果，这看上去其收敛速率不算太快，但这仅是一个内星，当网络变得较复杂时，其功效就更加明显了。一般情况下，科荷伦学习规则比内星学习规则能够提高训练速度 1 到 2 个数量级。

6.2 自组织竞争网络

6.2.1 网络结构

竞争网络由单层神经元网络组成，其输入节点与输出节点之间为全互联结。因为网络在学习中的竞争特性也表现在输出层上，所以在竞争网络中把输出层又称为竞争层，而与输入节点相连的权值及其输入合称为输入层。实际上，在竞争网络中，输入层和竞争层的加权输入和共用同一个激活函数，如图 6.3 所示。

竞争网络的激活函数为二值型{0,1}函数。

从网络的结构图中可以看出，自组织竞争网络的权值有两类：一类是输入节点 j 到 i 的权值 w_{ij} ($i = 1, 2, ..., s$; $j = 1, 2, ..., r$)，这些权值是通过训练可以被调整的；另一类是竞争层中互相抑制的权值 w_{ik} ($k = 1, 2, ..., s$)。这类权值是固定不变的，且它满足一定的分布关系，如距离相近的抑制强，距离远的抑制弱。另外，它们是一种对称权值，即有 $w_{ik} = w_{ki}$，同时相同神经元之间的权植起加强的作用，即满足 $w_{11} = w_{22} = ... = w_{kk} > 0$，而不同神经元之间的权值相互抑制，对于 $k \neq i$ 有 $w_{ij} < 0$。

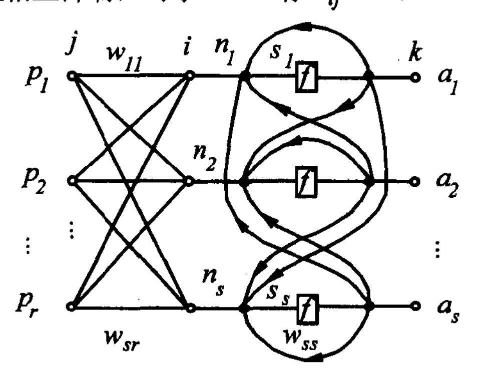


图 6.3　竞争网络结构图

下面来具体分析竞争网络的输出情况。

设网络的输入矢量为：$P = [\, p_1\, p_2 \ldots p_r \,]^T$；

对应网络的输出矢量为：$A = [\, a_1\, a_2 \ldots a_s \,]^T$。

由于竞争网络中含有两种权值，所以其激活函数的加权输入和也分为两部分：来自输入节点的加权输入和 N 与来自竞争层内互相抑制的加权输入和 G。具体地说，对于第 i 个神经元有：

1 ）来自输入节点的加权输入和为：

$$n_i = \sum_{j=1}^{r} w_{ij} \cdot p_j$$

2 ）来自竞争层内互相抑制的加权输入和为：

$$g_i = \sum_{k \in D} w_{ik} \cdot f(a_k)$$

这里 D 表示竞争层中含有神经元节点的某个区域，如果 D 表示的是整个竞争层，竞争后只能有一个神经元兴奋而获胜；如果竞争层被分成若干个区域，则竞争后每个区域可产生一个获胜者。

由于 g_i 与网络的输出值 a_k 有关，而输出值又是由网络竞争后的结果所决定的，所以 g_i 的值也是由竞争结果确定的。为了方便起见，下面以 D 为整个网络输出节点的情况来分析竞争层内互相抑制的加权输入和 g_i 的可能结果。

a)如果在竞争后，第 i 个节点“赢”了，则有：

$$a_k = 1, \quad k = i$$

而其他所有节点的输出均为零，即：

$$a_k = 0, \quad k = 1, 2, \ldots, s; \quad k \neq i$$

此时

$$g_i = \sum_{k=1}^{s} w_{ik} \cdot f(a_k) = w_{ii} > 0$$

b)如果在竞争后，第 i 个节点“输”了，而“赢”的节点为 l，则有：

$$a_k = 1, \quad k = l$$
$$a_k = 0, \quad k = 1, 2, \ldots, s; \quad k \neq 1$$

此时

$$g_i \sum_{k=1}^{s} w_{ik} \cdot f(a_k) = w_{il} < 0$$

所以对整个网络的加权输入总和有下式成立：

$s_l = n_l + w_{ll}$　　对于“赢”的节点 l

$s_i = n_i - |w_{il}|$　　对于所有”输“的节点 $i = 1, 2, \ldots, s, i \neq l$

由此可以看出，经过竞争后只有获胜的那个节点的加权输入总和为最大。竞争网络的输出为：

$$a_k = \begin{cases} 1 & s_k = \max(s_i, i = 1,2,\ldots,s) \\ 0 & \text{其他} \end{cases}$$

因为在权值的修正过程中只修正输入层中的权值w_{ij}，竞争层内的权值w_{ik}是固定不变的，它们对改善竞争的结果只起到了加强或削弱作用，即对获胜节点增加一个正值，使其更易获胜，对输出的节点增加一个负值，使其更不易获胜，而对改变节点竞争结果起决定性作用的还是输入层的加权和n_i，所以在判断竞争网络节点胜负的结果时，可直接采用n_i，即：

$$n_{\text{赢}} = \max\left(\sum_{j=1}^{r} w_{ij} p_j\right)$$

取偏差 B 为零是判定竞争网络获胜节点时的典型情况，偶而也采用下式进行竞争结果的判定：

$$n_{\text{赢}} = \max\left(\sum_{j=1}^{r} w_{ij} p_j + b\right), \quad -1 < b < 0$$

典型的 b 值取-0.95 。加上 b 值意味着取 $b = -|w_{il}|$ 这一最坏的情况。

通过上面分析，可以将竞争网络的工作原理总结如下：竞争网络的激活函数使加权输入和为最大的节点赢得输出为 1，而其他神经元的输出皆为 0。这个竞争过程可用 MATLAB 描述如下：

```
n = W*P;
[ S, Q ] = size (n);
x = n + b*ones (1, Q );
y = max (x);
for q = 1:Q
    z = find( x (:,q) == y(q));        % 找出最大加权输入和 y(q)所在的行;
```

```
    a(z(1),q) = 1;               % 令元素 a(z,q) = 1，其他值为零;
end
```

这个竞争过程的程序已被包含在竞争激活函数 **compet.m** 之中，可与其他函数一样简单的方式来调用它即可得到竞争网络的输出值:

```
A = compet ( W*P, B );
```

6.2.2 竞争学习规则

竞争网络在经过竞争而求得获胜节点后，则对与获胜节点相连的权值进行调整，调整权值的目的是为了使权值与其输入矢量之间的差别越来越小，从而使训练后的竞争网络的权值能够代表对应输入矢量的特征，把相似的输入矢量分成了同一类，并由输出来指示所代表的类别。

竞争网络修正权值的公式为:

$$\Delta w_{ij} = lr \cdot (p_j - w_{ij})$$

式中 lr 为学习速率，且 $0 < lr < 1$，一般的取值范围为 $0.01 \sim 0.3$；p_j 为经过归一化处理后的输入。

用 MATLAB 工具箱来实现上述公式的过程可以用内星学习规则:

```
A = compet ( W*P );
dW = learnis ( P, A, lr,W );
W = W + dW;
```

更省时地是采用科荷伦学习规则如下:

```
A = compet ( W*P );
i = find (A == 1 );
dW = learnis ( P, i, lr,W );
W = W + dW;
```

不论采用哪种学习方法，层中每个最接近输入矢量的神经元，通过每次权值调整而使权值矢量逐渐趋于这些输入矢量。从而竞争网络通过学习而识别了在网络输入端所出现的矢量，并将其分为某一类。

6.2.3 竞争网络的训练过程

弄懂网络的训练过程是为了更好地设计出网络。

因为只有与获胜节点相连的权值才能得到修正，通过其学习法则使修正后的权值更加接近其获胜输入矢量。结果是，获胜的节点对将来再次出现的相似矢量（能被偏差 b 所包容，或在偏差范围以内的），更加容易赢得该节点的胜利。而对于一个不同的矢量出现时，就更加不易取胜，但可能使其他某个节点获胜，归为另一类矢量群中。随着输入矢量的重复出现而不断地调整与胜者相连的权矢量，以使其更加接近于某一类输入矢量。最终，如果有足够的神经元节点，每一组输入矢量都能使某一节点的输出为 1 而聚为该类。通过重复训练，自组织竞争网络将所有输入矢量进行了分类。

所以竞争网络的学习和训练过程，实际上是对输入矢量的划分聚类过程，使得获胜节点与输入矢量之间的权矢量代表获胜输入矢量。

这样，当达到最大循环的值后，网络已重复多次训练了 P 中的所有矢量，训练结束后，对于用于训练的模式 P，其网络输出矢量中，其值为 1 的代表一种类型，而每类的典型模式值由该输出节点与输入节点相连的权矢量表示。

竞争网络的输入层节点 r 是由已知输入矢量决定的，但竞争层的神经元数 s 是由设计者确定的，它们代表输入矢量可能被划为的种类数，其值若被选得过少，则会出现有些输入矢量无法被分类的不良结果，但若被选得太大，竞争后可能有许多节点都被空闲，而且在网络竞争过程中还占用了大量的设计量和时间，在一定程度上造成了一定的浪费，所以一般情况下，可以根据输入矢量的维数及其估计，再适当地增加些数目来确定。

另外还要事先确定的参数有：学习速率和最大循环次数。竞争网络的训练是在达到最大循环次数后停止，这个数一般可取输入矢量数组的 15~20 倍，即使每组输入矢量能够在网络重复出现 15~20 次。

竞争网络的权值要进行随机归一化的初始化处理，这个过程在 MATLAB 中用函数 **randnr.m** 实现：

```
W = randnr ( S, R );
```

然后网络则可以进入竞争以及权值的调整阶段。

网络的训练全过程完全由计算机去做，工具箱中的竞争网络训练函数为 **trainc.m**，调用时所需要的参数为：初始权矩阵 W，输入矩阵 P 和具有三个训练参数的行矢量：显示循环的频率 disp_freq，最大的训练次数 max_cycle 以及学习速率 lr，它的用法如下：

```
TP = [ disp_freq    max_cycle    lr ];
W = trainc( W, P, TP );
```

竞争网络比较适合用于具有大批相似数组的分类问题。下面给出例题来说明竞争网络的功效。

【例 6.4】对下列模式 P 进行分类辨识。

```
P = [ 0.7071 0.6402 0.000 -0.1961 0.1961 -0.9285 -0.8762 -0.8192 ;
      0.7071 0.7682 -1.000 -0.9806 -0.9806 0.3714 0.4819 0.5735 ];
```

解:

输入模式 P 已经为归一化处理后的数据。对于网络结构，我们取 S = 4；根据:

```
W0 = randnr(S,R) ;
```

随机取一组初始矩阵为:

```
W0 = [ -0.3347 -0.9413;
       -0.5466 -0.8374;
       -0.8690 -0.4948;
       0.4611 -0.8874];
```

另取:

```
lr = 0.05;
max_epoch = 320;
```

这里取了输入数据的 20 倍。最大循环数一般应根据输入数据的多少来决定。

（1）训练竞争

为了能够更清楚地理解训练竞争过程，我们仍可以通过图解法来解释。和感知器的分类方式类似，二值型分类的实质是通过将输入矢量空间进行分割而达到分类目的。在此，我们用横纵坐标分别表示 p_1 和 p_2 矢量。与感知器的不同的是，竞争网络所用的矢量模为 1，所以矢量的变化，在坐标上形成的轨迹为以原点为中心的单位圆。网络竞争的目的，是使权值 W 经过竞争后逐渐移动到能够代表输入矢量类别的点上（也处于单位圆上）.

在训练过程中，当第一次出现输入矢量比如 P^1 时，有 $W_{ij}=p^1$，$l \in s, j=1,2,\dots,r$。但当 P^2 出现时，若也具有与 W_{lj} 相似的特性。则通过对 W_{lj} 的修正，使 W_{lj} 倾向 P^2，当下一次 P^1 再次输入时，W_{lj} 又被修正得移向 P^1。这样，经过多次训练后，W_{lj} 所得的结果为几个相同类型输入模式的平均值，而当一个不同类型的模式输入，将使竞争网络中的其他节点获胜而得到一个新的权矩阵。如此重复，可以将所有的不同类型的模式都聚集到相同的权矩阵下，每个权矩阵代表一种类型，而输出矢量中的 1 的列位置则指出与此节点相连的权矢量所代表的输入矢量组的位置。图 6.5 给出最后的训练结果，图中权矢量位置是用“o”来表示的。

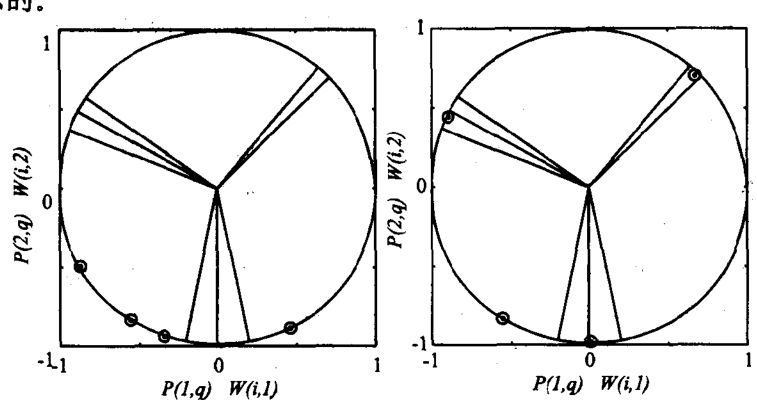


图 6.4 竞争前的权矢量位置图　　图 6.5 竞争结束后的权矢量位置图

本例题经过 320 次循环后得到最后结果为:

W=[-0.0460 -0.9863;
-0.5466 -0.8374;
-0.8748 0.4747;
0.6684 0.7180];

在输入模式 P 作用下的网络输出为:

A=[00111000; —1类 (第3、4、5列输入矢量组属于同一类)
00000000; — 空缺
00000111; —3类 (第6、7、8列输入矢量组属于同一类)
11000000]; —4类 (第1、2列输入矢量组属于同一类)

将 W 值与前面 $W0$ 相比较可以看出，第二行的值与初始 $W0$ 值完全相同，即在整个竞争训练中，W_{2j} 从未获得过胜利，所以其权矩阵一次也没有得到过修正，这从 A 的输出值上看得很清楚：当 P^1、P^2 输入时，输出节点 4 获胜，即它们属于 W_{4j} 类；W_{4j} 可代表它们的值，P^3、P^4 和 P^5 属第一类，W_{1j} 是他们三个的代表，P^6、P^7 和 P^8 同属第 3 类，W_{3j} 反映了它们的特点。

（2）竞争学习网络的局限性

竞争网络适用于当具有典型聚类特性的大量数据的辨识，但当遇到大量的具有概率分布的输入矢量时，竞争网络就无能为力了，这时可以采用科荷伦网络来解决。

6.3 科荷伦自组织映射网络

神经细胞模型中还存在着一种细胞聚类的功能柱。它是由多个细胞聚合而成的，在接受外界刺激后，它们会自动形成。一个功能柱中的细胞完成同一种功能。

生物细胞中的这些现象在科荷伦网络模型中有所反映。当外界输入不同的样本到科荷伦自组织映射网络中，一开始时输入样本引起输出兴奋的位置各不相同，但通过网络自组织后会形成一些输出群，它们分别代表了输入样本的分布，反映了输入样本的图形分布特征，所以科荷伦网络常常被称为特性图。

科荷伦网络使输入样本通过竞争学习后，功能相同的输入靠得比较近，不同的分得比较开，以此将一些无规则的输入自动排开，在联接权的调整过程中，使权的分布与输入样本的概率密度分布相似。所以科荷伦网络可以作为一种样本特征检测器，在样本排序、样本分类以及样本检测方面有广泛地应用。

一般可以这样说，科荷伦网络的权矢量收敛到所代表的输入矢量的平均值，它反映了输入数据的统计特性。再扩大一点，如果说一般的竞争学习网络能够训练识别出输入矢量的点特征，那么科荷伦网络能够表现出输入矢量在线上或平面上的分布特征。

当随机样本输入到科荷伦网络时，如果样本足够多，那么在权值分布上可近似于输入随机样本的概率密度分布，在输出神经元上也反映了这种分布，即概率大的样本集中在输

出空间的某一个区域，如果输入的样本有几种分布类型，则它们各自会根据其概率分布集中到输出空间的各个不同的区域。每一个区域代表同一类的样本，这个区域可逐步缩少，使区域的划分越来越明显。在这种情况下，不论输入样本是多少维的，都可投影到低维的数据空间的某个区域上。这种形式也称为数据压缩。同时，如果高维空间中比较相近的样本，则在低维空间中的投影也比较相近，这样就可以从中取出样本空间中较多的信息。遗憾的是，网络在高维映射到低维时会出畸变，且压缩比越大，畸变越大；另外网络要求的输入节点数很大，因而科荷伦网络比其他人工神经网络（如网络 BP 网络）的规模要大。

6.3.1 科荷伦网络拓扑结构

科荷伦网络结构也是两层：输入层和竞争层。与基本竞争网络不同之处是其竞争层可以由一维或二维网络矩阵方式组成，且权值修正的策略也不同。

1）一维网络结构与基本竞争学习网络相同；

2）二维网络结构，如图 6.6 所示，网络上层有输出节点 s 个，按二维形式排成一个节点矩阵，输入节点处于下方，有 r 个矢量，即 r 个节点，所有输入节点到所有输出节点之间都有权值连接，而且在二维平面上的输出节点相互间也可能是局部连接的。

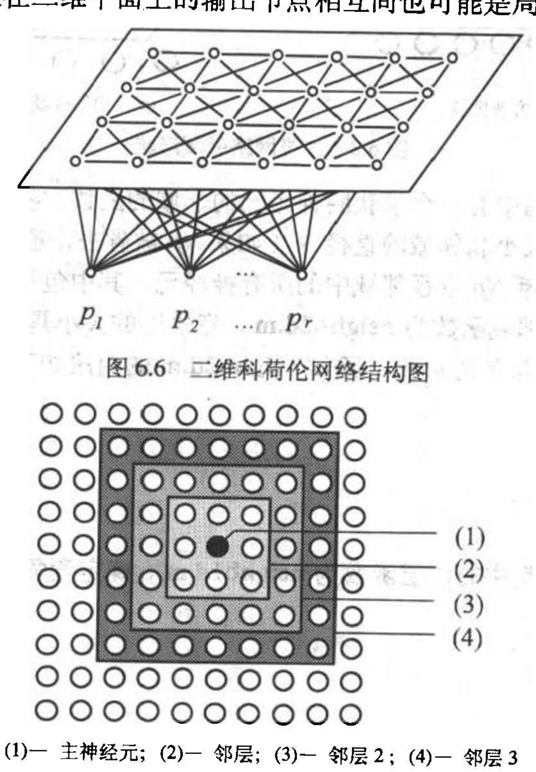


图 6.6 二维科荷伦网络结构图

(1)— 主神经元；(2)— 邻层；(3)— 邻层 2；(4)— 邻层 3

图 6.7 二维神经元层示意图

科荷伦网络的激活函数为二值型函数。一般情况下 b 值固定，其学习方法与普通的竞争学习算法相同。在竞争层中，每个神经元都有自己的邻域，图 6.7 为一个在二维层中的主神经元。主神经元具有在其周围增加直径的邻域。一个直径为 1 的邻域包括主神经及它

的直接周围神经元所组成的区域；直径为 2 的邻域包括直径 1 的神经元以及它们的邻域。图中主神经元的位置是通过从左上端第一列开始顺序从左到右，从上到下找到的。如图中的 10 × 10 神经元层，其主神经元位于 46 。

特性图的激活函数也是二值型函数，同竞争网络一样可以取偏置 b 为零或固定为一常数。竞争层的竞争结果，不仅使加权输入和为最大值者获胜而输出为 1 ，同时也使获胜节点周围的邻域也同时输出为 1 。另外，在权值调整的方式上，特性图网络不仅调整与获胜节点相连的权值，而且对获胜节点邻域节点的权值也进行调整，即使其周围 D_k 的区域内神经元在不同程度上也得到兴奋，在 D_k 以外的神经元都被抑制，这个 D_k 区域可以是以获胜节点为中心的正方形，也可以为六角形，如图 6.8 所示。对于一维输出， D_k 则为以 k 为中心的下上邻点。

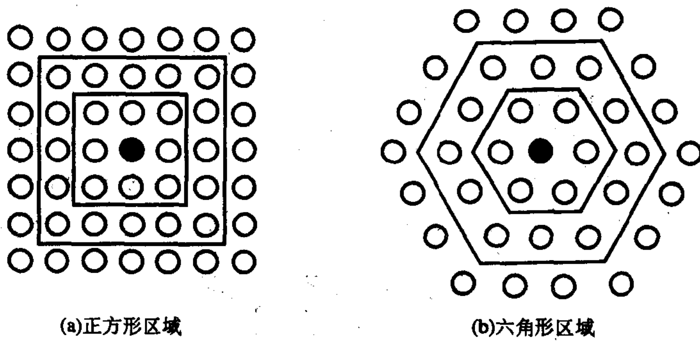


图 6.8　二维网络邻域形状

在 MATLAB 工具箱中有一个求获胜神经元的邻域的函数，它所需要的参数为：获胜神经元位置 i ，竞争层的大小和邻域的直径 N （如果 Np 被省去，意味着直径为 1 ），邻域函数返回一个行矢量，包括 Np 直径邻域中的所有神经元，其中包括主神经元本身在内。

在二维竞争层中，邻域函数为 **neighb2d.m** ，竞争层的大小是其宽度 X 和高度 Y ，层中神经元总数 S 应当是 X 与 Y 的乘积。函数 **neighb2d.m** 的用法如下：

```
Np = [ X Y ];
in = neighb2d (i,Np,N);
```

对于一维竞争层，其中的邻层函数为 **neighb1d.m** ，确定竞争层大小的参数就是神经元数 S ，即

```
Np = [ S ];
in = neighb1d (i,Np,N);
```

除了正方形以外，我们可以用函数 **neighb2d.m** 产生一个具有任意形状的邻层。此时，对新邻域函数的唯一限制是变量列表必须与上述两个函数相同。不过，邻层函数的行矢量可以包含任何常数。

6.3.2 网络的训练过程

科荷伦网络在训练开始时和普通的竞争网络一样，其输入节点竞争的胜利者代表某类模式。然后定义获胜节点的邻域节点，即以获胜节点为中心的某一半径内的所有节点，并对与其相似的权矩阵进行调整。随着训练的继续进行，获胜节点 k 的半径将逐渐变小，直到最后只包含获胜节点 k 本身。也就是说，在训练的初始阶段，不但对获胜的节点作权值的调整，而且对其周围较大范围内的几何邻接节点也作相应的调整，而随着训练过程的进行，与获胜输出节点相连的权矩阵就越来越接近其所代表的模式类，此时，需要对获胜节点进行较细致的权矩阵调整。同时，只对其几何邻接较接近的节点进行相应的调整。这样，在训练结束后，几何上相近的输出节点所连接的权矢量既有联系(即类似性)，又相互有区别，保证了对于某一类输入模式、获胜节点能作出最大的响应。而相邻节点作出较少响应。几何上相邻的节点代表特征上相似的模式类别。

特性图的初始权值一般被设置得很小，例如初始化一个特性图用 R 个输入和 S 个神经元的过程可为：

W = randnr(S,R)*0.1；

由上可知，特性图不同于常规竞争学习网络那样修正其权值，除了修正获胜权值外，特性图还修正它的邻域权值。结果是邻域的神经元也逐渐趋于相似的权矢量，并对相似的输入矢量作出响应。

训练设计步骤（适用于输入矢量 P 具有某种概率分布的数组）：

（1）初始化

1）由输入矢量确定网络结构：[R,Q] = size(P);

2）设置网络竞争层神经元节点：一维 S 或二维的宽 X 和高 Y， S = X*Y；

3）将输入模式 P 作归一化处理： P = normc(P);

4）归一随机化处理初始权值： W = rands(S,R)*0.1； 并设置：

5）最大循环次数（此数根据输入数据的数目而乘一个倍数所得）： max_cycle;

6）基本学习速率 lr: 一般取 0.01 ~ 0.3，视具体情况而定;

7）最大邻层数 max_neighb: 一维 max_neighb = S - 1;

二维 max_neighb = max ([X Y]) - 1;

（2）循环训练

for cycle = 1: max_cycle

1）学习速率是线性下降的：

```
x = cycle/max_cycle;
LR = (1 - x )*lr;
```

这使得学习速率随着循环次数的增加，从 lr*(max_cycle-1)/max_cycle 逐渐降至 0；

2 ）邻层数也是递减的:

```
n = max([ceil(max_neighb*(1-x*4)) 1]);
```

3 ）计算输入矢量加权和，并通过竞争求出获胜节点:

```
A = compet(W*P);
```

4 ）根据获胜节点求出相邻层（以二维为例），并进行权值修正:

```
i = find(A ===1);
in = neighb2d(i,[X Y],n);
dW = learnk(W,P,in,LR);
W = W + dW;
```

（3）输出或显示结果

在训练过程中，神经元参与彼此的竞争活动，而具有最大输出的神经元节点是获胜者。该获胜节点具有抑制其他竞争者和激活其近邻节点的能力，但是只有获胜节点才允许有输出，也只有获胜者和其邻近节点的权值允许被调节。获胜者的近邻节点的范围在训练过程中是可变的。在训练开始时，一般将邻近范围取得较大，随着训练的进行，其邻近范围逐渐缩小。因为只有获胜节点是输入图形的最佳匹配，所以说科荷伦网络模仿了输入图形的分布，或者说该网络能提取输入图形的特征，把输入图形特征相似地分到一类，由某一获胜节点表示。

MATLAB 工具箱中用于训练设计科荷伦网络权矢量的函数为 **trainfm.m**，它能执行上述的训练步骤的全过程，仅需要给它输入 4 个训练参数；显示频率 disp_freq，最大训练周期 max_cycle，学习速率 lr 以及最大邻域数 max_neighb :

```
TF = [disp_freq max_cycle lr max_neighb ];
W = trainfm ( W, P, TF );
```

最大邻域数应设置为层的最大直径数减去 1 。例如，对一维特性图 max_neighb 应为 S - 1；对二维特性图，其层神经元宽为 X 高为 Y 时，max_neighb 应当等于两者中的较大值减去 1 。不论哪个神经元被选为主神经元，最大邻域数总是包括了层中的所有神经元。

函数 **trainfm.m** 的训练开始于学习速度 lr 和最大邻域 max_neighb，然后，其学习速率线性地递减，以致于最后的训练周期里的学习速率为 0 。邻域数也是线性地减少，但在达到四分之一训练周期后，其值保持为最大值 1 直到最终训练结束。

这种安排允许输出神经元在初始阶段的活动范围较大，并使其全体都趋于输入矢量出现的区域。然后，随着邻层的减少至 1 ，图形趋于所出现的输入矢量的自身。一旦邻域大小为 1 ，图形则已被很好地排了序。学习速率也是在一个较长的时间里进行衰减，以给神经元足够的时间展开。

如果输入矢量是以遍历整个输入空间的概率出现，特性图训练的最终结果将是以接近

于输入矢量之间等距离的位置排列；如果输入矢量是以遍历输入空间的变化频率出现，特性图将趋于将神经元定位为一个正比于输入矢量频率的面积。由此，特性图通过被训练，能够学习输入矢量的类型与其贡献的大小，而将其分类。

给定一个特性图的权矩阵 W，它的邻域函数 F（作为一个字符串），以及竞争层大小 Np，可以用函数 **plotmap.m** 画出特性图。

下面给出有关特性图的应用实例，从中可以看出特性图是如何被训练设计的。

【例 6.5】有 100 个输入矢量均匀地分布在单位圆从 0 度到 90 度的圆周上，试设计训练一个特性图将其取而代之。

解:

根据题意，输入矢量可定义可为:

```
angles = 0:0.5*pi/99:0.5*pi;
P = [sin(angles); cos(angles)];
```

输入矢量如图 6.9 所示。

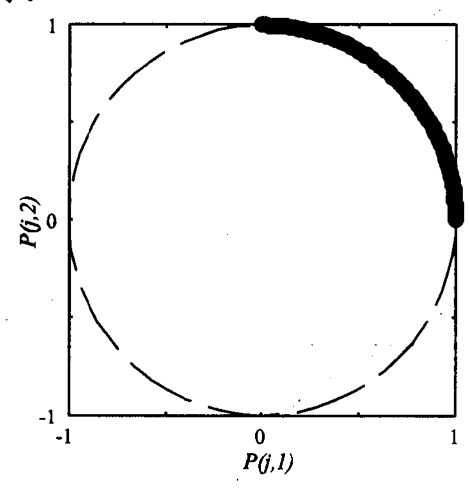


图 6.9　输入矢量图

所设计科荷伦网络为一维特性图，取竞争层神经元数为 20，所以取 **neighb1d.m** 函数来训练，并用函数 **trainfm.m** 进行训练，特性图将被训练 400 次，初始学习速率为 0.3，最大邻域为 S - 1，训练程序如下:

```
[R,Q] = size(P);
S = 20;
W0 = rands(S,R)*0.1;
disp_freq = 4;
max_cycle = 400;
lr = 0.3;
max_neighb = S - 1;
```

```
TF =   [ disp_freq max_cycle lr max_neighb ];
W = trainfm ( W0, P,'neighb1d',S,TF );
```

图 6.10 给出用函数 plotmap(W0, 'neighb1d',S)画出的原始随机权值的特性图，其中，每个神经元的权矢量被表示为单位圆中的一个点，每个点通过一条线与其邻域点相连，小的随机初始权矢量导致在圆中心附近无序的图形。

当训练开始后，权矢量开始一起朝着输入矢量的位置移动，并随着邻域范围的减少，权矢量之间也逐渐变得有序起来。随着训练的继续，神经元层的权矢量变得越来越有序，且逐渐覆盖了整个输入矢量的位置。典型的训练 50 次和 100 次的结果图如图 6.11 和图 6.12 所示。

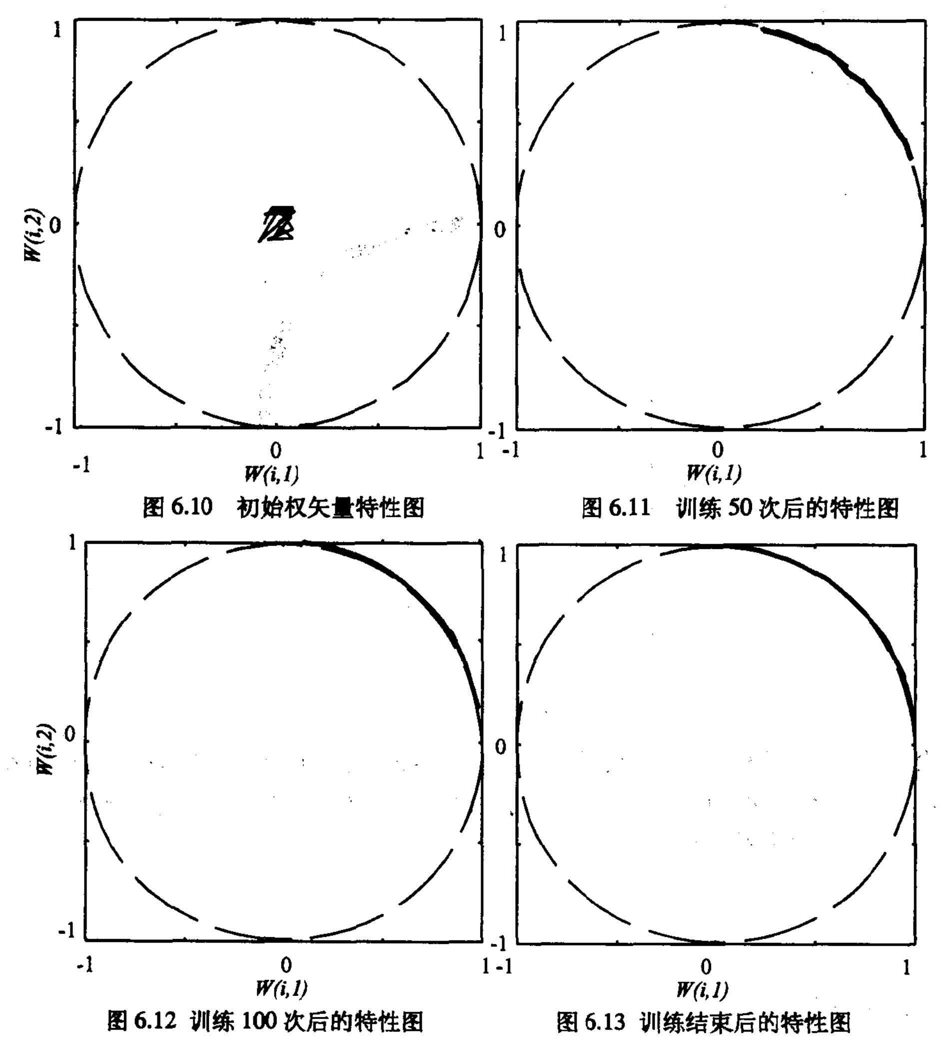


图 6.10　初始权矢量特性图

图 6.11　训练 50 次后的特性图

图 6.12 训练 100 次后的特性图

图 6.13 训练结束后的特性图

在训练 400 次后，神经元层已基本调整好了自身的权矢量，能够使每一个神经元响应输入的矢量所在的一个小区域，并通过邻域完成了用 20 个神经元联系了 100 个元素的特性图功能。从最后的训练结果图 6.13 可以看出，由于输入矢量是均匀分布，获胜节点输出 1

的分布，也是随着输入矢量出现的先后顺序逐渐均匀变化的。经过网络训练和权矢量调整，随机的初始权矢量逐渐向第一象限的单位圆周上移动，最终 20 个权矢量均匀地分布在四分之一圆周上，替代了 100 个输入矢量.

【例 6.6】有 2 000 个二元随机输入样本产生一个平方面积为 0.5 • 0.5 的覆盖区域，试用科荷伦网络取代之。

解:

根据题意可将输入矢量用下列函数产生:

```
P = rands(2, 2000)*0.5;
```

但是，特性图中的输入矢量必须是单位长度，而现在所有的输入矢量的幅值小于 1 ，所以可以通过对输入矢量再加上第三个元素，使之与原始两个元素组成的新的输入矢量具有单位 1 的模值，这个过程可以用函数 **pnormc.m** 来完成:

```
P = pnormc(P);
```

图 6.14 是由归一化后的输入矢量的前两个元素产生的输入图形，也是用我们感兴趣的输入矢量画出的输入矢量平面图. 增加的一个元素已使输入变成了三维空间，但我们所感兴趣的仍然是二维平面里的输入矢量.

下面进行有关参数的选择和初始化工作:

```
X = 5; Y = 5; S = X*Y;
[R,Q] = size(P);
W0 = rands(S,R)*0.1;
plotmap(W0,'neighb2d',[X Y]);
```

一个典型的初始权矢量图如图 6.15 所示。

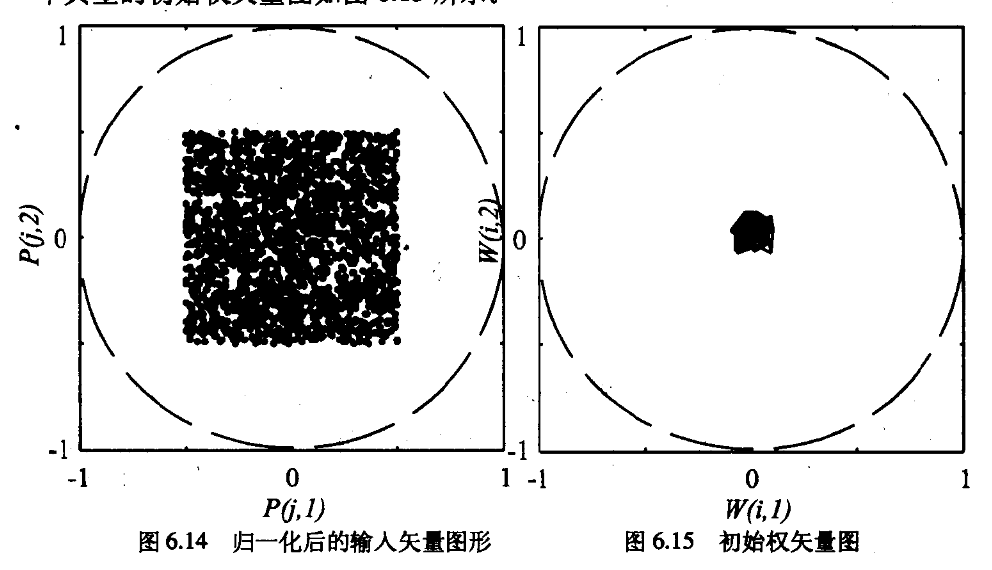


图 6.14 归一化后的输入矢量图形　　图 6.15 初始权矢量图

若取训练次数为 4 000 次，训练主程序如下:

```
disp_freq = 25;
max_cycle = 4000;
lr = 0.025;
max_neighb = max([X Y]) - 1;
TF =    [ disp_freq max_cycle lr max_neighb ];
W = trainfm ( W0, P, 'neighb2d',[X Y],TF );
```

随着训练的进行，所有神经元权矢量均朝着输入矢量的中心移动，并逐渐地在输入矢量的位置上排序，图 6.16 和图 6.17 分别给出了在 800 次和 4 000 次训练后的特性图。

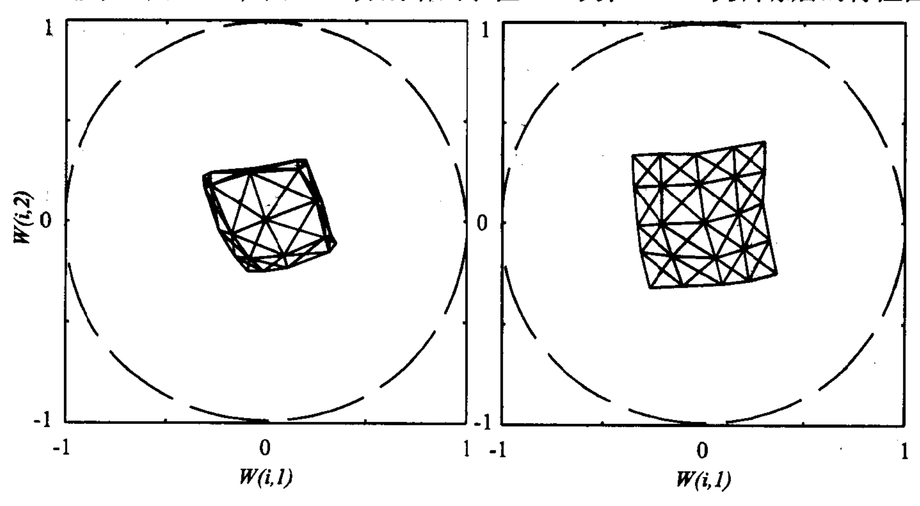


图 6.16　训练 800 次后的特性图　　图 6.17　训练 4 000 次后的特性图

注意，最后的图形并不特别光滑. 当然可以替换其他更大的训练次数来更进一步训练网络。有时特性图在其训练中也会出现麻烦，比如产生来会摆动而使其扩展延伸停止，这时可以通过增加训练次数来加以避免. 学习速率太小也可能是其原因之一，不过太大的学习速率可能导致不稳定。实际上，几乎所有可能出现的问题，均能够通过采用更多的训练次数来加以解决。

6.4 对传网络

对传网络(Counter Propagation Network, 简称 CPN)是美国学者 Hechi-Nielson 在 1987 年首次提出的。从结构上看， CPN 是一种层次结构的网络，实际上， CPN 是把两种著名的网络算法；科荷伦自组织映射理论与格劳斯贝格外星算法组合起来而形成的网络。

6.4.1 网络结构

CPN 网络为两层结构：第一层为科荷伦层，采用无指导的训练方法对输入数据进行自

组织竞争的分类或压缩，第二层称为格劳斯贝格层。

第一层的激活函数为二值型硬函数，而第二层为线性激活函数。

1）对于科荷伦层的输出有：

$$K = F1*(W1*P + B1)$$

2）对于格劳斯贝格层，具有目标矢量 G，此时，从前层输出的 K 为它的输入，输出为：

$$G = F2*(W2*K + B2) = W2*K + B2$$

CPN 的网络结构如图 6.18 所示。

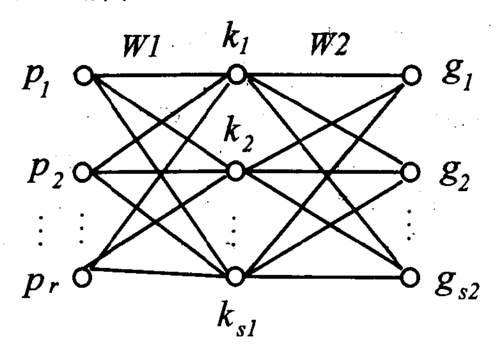


图 6.18　对传网络结构图

6.4.2 学习法则

1）在科荷伦层，通过竞争对获胜节点采用科荷伦规则调整与其相连的权矢量：

$$\Delta W1 = lr1*(P - W1)$$

2）在格劳斯贝格层，对与在科荷伦层输出为 1 的输入相连的权值进行如下的调整：

$$\Delta W2 = lr2*(G - W2)$$

这又是外星规则的变形。以此来迫使权矢量逼近与输入 1 对应的输出目标矢量。

6.4.3 训练过程

1）初始化：

归一化处理输入、目标矢量 P 和 G；

对权矢量 $W1$ 和 $W2$ 进行归一化随机取值；

选取最大循环次数、学习速率 $lr1$ 和 $lr2$。

2）科荷伦层的无指导训练过程：

重复对输入的样本进行竞争计算，对获胜的科荷伦层获胜节点按科荷伦法对与其连接的权矢量进行修正；

3）格劳斯贝格层有指导的训练过程：

寻找层输入为 1 的节点，并对与该节点相连的权矢量进行修正；

4）检查最大循环的数是否达到，是，则停止训练，否，则转入 2）。

经过充分训练后的 CPN 可使其科荷伦层的权矢量收敛到相似输入矢量的平均值，而使格劳斯贝格层权向量收敛到目标矢量的平均值。

当 CPN 训练后工作时，只要对网络输入一矢量 X，则在科荷伦层经过竞争后产生获胜节点，并在格劳斯贝格层使获胜节点所产生的信息向前传送，在输出端得到输出矢量 Y，这种由矢量 X 得到矢量 Y 的过程有时也称为异联想，更广泛地说，它实现了一种计算过程。

当训练 CPN 使其格劳斯贝格层的目标矢量 G 等于科荷伦层的输入矢量 P 时，则可实现数据压缩。具体做法是：首先是训练 CPN 使其 $G=P$，然后，将输入数据输入 CPN，在科荷伦层输出得到 0，1 数据，这些数据为输入的压缩码。解码时，将在科荷伦层压缩的 0，1 码送入格劳斯贝格层，在其输出端对应得到解压缩的矢量。

实际上从格劳斯贝格层输出端得到的仅是同类矢量的平均值，并不是精确的原始矢量，它只反映了输入矢量的统计特性。若采用 BP 网络则能够得到更精确的信息，所以对于某些要求精确映射的问题，使用 CPN 不太合适，不过 CPN 先把输入矢量聚类，然后再用格劳斯贝格层进行监督式训练，这种处理方式在不少场合下是适用的，并且节省了大量的时间，它的最突出的优点是将监督式和无监督式的训练方法有机的结合起来了，从而提高了训练速度。

6.5 自适应共振理论

自适应共振理论(Adaptive Resonance Theory,简称 ART)是美国波士顿大学的卡潘特(A. Carpenter)和格劳斯贝格提出的。这种网络具有两种形式：ART1 处理双极性(或二进制)数据，而 ART2 处理连续数据。

从前面多种网络对样本数据训练的过程中已经看到，不论是监督式或是无监督式的训练，都会出现网络对新模式的学习。伴随着对已学习过的模式的部分甚至全部的忘却，在监督式的训练情况下，我们通过对网络成千上万次地反复输入样本的训练，使网络逐渐达到稳定的记忆。而在无监督式的训练情况下，对新的数据的学习将会产生对某种已记忆的典型矢量的修改，造成对已学习数据的部分忘却，控制不好将会使所记忆的矢量来回波动而变成没有代表意义。所以在神经网络的训练过程中，我们时刻面临着新知识的学习记忆和对旧知识的退化忘却这个问题。我们的希望是，能够学会新的知识，而同时对已学过的知识毫无不利影响。但是在输入矢量特别大的情况下，很难达到这种愿望。一般情况下，我们只能在新旧知识的取舍上进行某种折衷，最大可能地接受新的知识而较少地影响原有的知识。

ART 网络较好地解决了上述的问题。网络和算法有较大的灵活性，以适应新输入的模式，同时极力避免对网络先前所学习过的模式的修改。它的记忆容量可以随样本的增加而自动增加，这就可以在不破坏原记忆样本的情况下学习新的样本。

ART 网络的主要应用方向集中在图像处理、语声处理、模式识别、控制系统、知识处理、雷达图像分类以及语声感知等领域。

6.5.1 ART1 网络结构

ART1 的一种网络结构如图 6.18 所示。主要由两层神经元以及一些控制信号相互结合而成。为了方便起见，人们一般把从输入矢量 P 到输出矢量 A 的称为识别层或 R(R－recognition)层，而把输出 A 作为输入再返回输入的层称为比较层，或 C(C- Comparison)层。

从层结构上讲，R 层为一个竞争网络；C 层为一个格劳斯贝格的外星网络。

由结构图 6.19 可以看出，R 层和 C 层分别接受来自三个方面的信号：

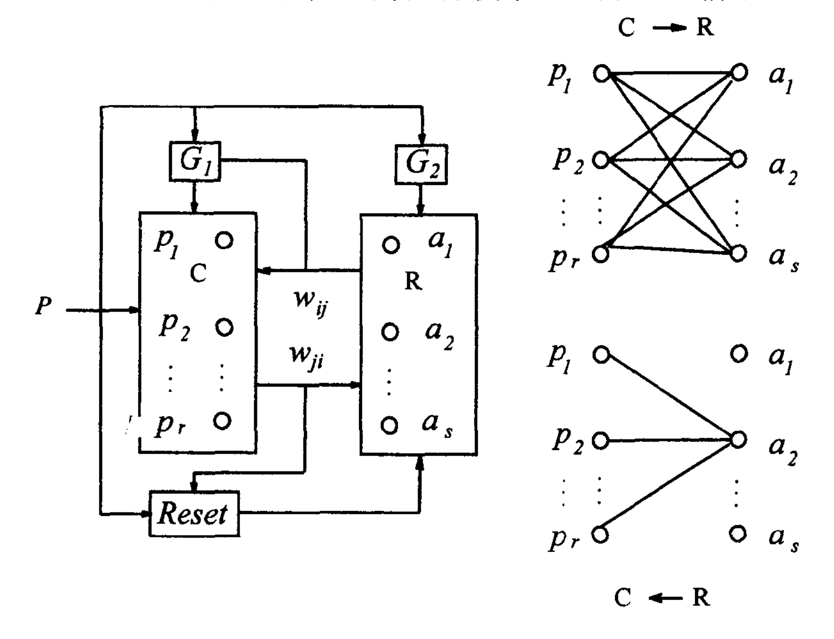


图 6.19 ART1 系统的一种结构图

R 层：

①加权输入经过竞争后的所产生的输出 Y；

②控制信号 $G2$;

③复原信号 Reset 。

C 层：

①输入矢量 P;

② R 层的返回信号;

③控制信号 $G1$ 。

控制信号 $G1$ ，$G2$ 和 Reset 的作用如下：

$G1$：

设输入矢量的各元素的逻辑“或”为 P_0，R 层输出元素的逻辑“或”为 A_0，则有 $G1 = P_0\bar{A}_0$。换句话说，只有当 R 层输出元素全为 0，而输入元素不全为零时，$G1 = 1$，其他情况下，$G1 = 0$ 。

$G2$:

等于输入元素的逻辑“或”，即只有在输入元素全为0时，$G2=0$，否则 $G2=1$。

Reset:

Reset 平时等于1，而当测试输入矢量与 R 层权值点各与预先设想的相似度 B 之和小于零时，以及在 R 层竞争获胜节点无效，表示此次选择的模式代表不能满足要求时，则发生 Reset = 0 的信号。

R 层输出有 s 个节点，它代表了 Q 组输入模式的分类，该节点数是能够动态地增长，以满足设立新模式的需要。ART 网络作为模式分类器，其工作过程如下：

网络接受输入矢量，并按一定的规则来确定这个新输入是否属于网络中已记忆的模式类别，其判别的标准为新输入模式与所有已记忆模式之间的相似程度。按照事先设定的相似度来考察它们，从而决定对新输入矢量采取的处理方式。对大于预定相似度的输入模式，选择其中最为相似的记忆模式作为该输入的代表类，并按一定的规则修正相似权矢量以使该类权值更加接近于新输入模式。若该输入模式不与现记忆的任何类别相似，则在网络中设立一个新的模式，用以代表和记忆新模式，并将其归为已有的代表类别，成为 R 层的一个新的输出节点，作为以后可能输入的代表模式。

在每一次网络接受新的输入时，都伴随着上面的运行过程，以决定本次输入模式是否应归并到已有的记忆类别中，归并到哪一类，还是应该新设立记忆节点。若已经确定输入模式与某一记忆类别相似，则调整与该类节点相连的权矢量，使以后该模式再次输入时，能够获得更大的相似度。对于其他权矢量，则不作任何变动。当设立一个新模式类时，同时建立与该模式相连的权矢量，以便记忆该模式并参与以后模式输入的运行过程。

6.5.2 ART1 的运行过程

ART 网络最主要的特点是将网络的训练与工作运行过程有机地结合在一起，一般 ART1 的工作过程都分为下面三个阶段：

（1）自 C 层流向 R 层的识别阶段

在输入一个模式 P 之前，C 层输入端的值均为0，使 $G2=0, G1=0$，而 Reset = 1，所以 R 层处于可运行的等待方式，因 R 层输入均为0，令 P 输入可运行。

当输入矢量 P 信号由左向右根据竞争层功能，首先计算输入加权总和，然后进行竞争，求出获胜竞争并进入比较阶段。

（2）自 R 层流向 C 层的比较阶段

一旦 R 层有1输出，同时令 $G1=0$，从而使 C 层输入悬空等待。R 层输入所得输出信号返回 C 层，这将与 R 层输出为1的节点相连的外星权值 W_{ji} 相乘获得外星加权总和，加上预先设置的相似度 B，这个权值是等于竞争网络中权值的 W_{ij} 的转置，R 层流向 C 层的最终端输出值，也是在 C 端获得的临时输入值为 C 外星网络饱和函数的输出：

$$C = F(W^*P + B)$$

若 C 值大于0，使 $G2=1$，同时置 Reset = 1，致使 R 层运算结果有效，表明竞争结果正确，输入矢量 P 也找到同类，此时可使竞争层权值得以修正：

$$\Delta W = lr * (P - \Delta W); \quad W = W + \Delta W;$$

而大于0的C值，也使 $G1 = 1$，加上不为零的 Y，使C层进入输入下一个新的状态。若C值等于0，使 $G2 = 0$，同时发出Reset = 0信号，从而使R层此次结果无效，进而输入搜索阶段。与此同时，置R层输出 $Y = 0$，并使R层获胜节点在后面的搜索阶段不能再获胜利。另一方面，C = 0与 $Y = 0$ 使C层处于仍输入现输入 P 的状态，而不输入新输入 P。

（3）搜索阶段

由 Reset 信号置获胜节点无效开始，网络进入搜索阶段。一旦在网络输入端出现阶段输入 P，则使 $G1 = 1$ 和 $G2 = 0$ 而使C层进入运行阶段。因此，整个网络又重新进入了识别和比较阶段，并可获得新的获胜节点。又因 Reset = 0，所以，前面的获胜节点不准许再参加竞争，这样重复直至搜索到某一个获胜节点 k 按外星网络返回的权矢量与输入 P 的相似度达到满意的要求为止。并把 P 归类为R层第 k 个节点所连接权矢量的类别中，即按一定的方式修改与第 k 个节点相连的内、外星权矢量（如内星采用科荷伦规则，外星权矢量为所求内星权矢量的转置）这样使网络以后再遇到 P 或 P 的相似模式时，R层的 k 节点能更快地使其获得竞争的胜利。

若搜索了所有的R层输入节点而没有找到与输入矢量 P 充分接近的外星权矢量，则增设一个新的R层节点以代表输入 P 或与 P 相近的权矢量。

以上三个阶段表示了 ART1 对一次外界输入网络的运行过程。当外界输入 P 与所激活的外星权矢量充分相似时，网络则发生“共振”，运行过程结果，否则的话，就进行特别的处理，直到“共振”的现象发生时对本次输入的训练过程才最终结束，然后置 Reset = 1，释放所有被 Reset 信号封住的节点，使之又可以对新的输入模式开始以上的训练过程。

R层和C层运行的最终有效性是根据各层的“2/3规则”来决定的。具体地说，对R层来说，其层运行竞争结果的有效性是由 $G2$ 与 Reset 两信号是否一致来决定有效或无效；对C层来说，控制信号 $G1$ 与R层返回信号是同时为0，或同时为1时，决定着该层输出矢量是原先输入矢量 P 还是下一个新输入矢量，表6.1表示一种可能的信号规则。

表 6.1　ART 网络运行的控制信号

R层

Reset	1	1	0	0
$G2$	0	1	0	0
R层	运行工作	结果有效	结果无效	锁住获胜节点

C层

R层输出Y	0	1	1	0
$G1$	1	0	1	0
输入 P	P 输入运行	输入悬空等待	输入下一个新 P	仍输入旧 P

类似上述 ART1 网络的工作过程已经由 MATLAB 编成了名为 **sima1.m** 的程序。它所需要的输入变量有4个：被学习的权值 $W1$，输入矢量 P，相似度 r 以及打印频率 pf。程序的输出为训练出的最新权矩阵 W，由训练权值所返回的网络输出矢量 $A1$，以及在竞争层所获胜的节点。

值得注意的是，对 ART1 网络进行训练时，初始权矢量 W 必须全取1。因为采用的是

0, 1 输入，对输入矢量不进行归一化处理，但所采用的相似度是将竞争获胜后返回到输入层的矢量 $A1$ 与输入矢量 P 进行逻辑“与”处理，然后用两者元素之和的差值进行比较。当相似度 $r=1$ 时，表示 $A1$ 与 P 在相同输入元素上具有相同数字 1 的数目相等，这也是两个矢量为完全相同的情况。随着 r 的减小，有差异的相似输入分量就有更多的可能性被分成同类。工作时，设计者只要编写一个自己的应用程序对函数 **sima1.m** 进行调用，即可求得 ART1 网络最后对输入矢量自动分类的结果。下面给出对一个具有 8 组 13 个元素的二进制输入矢量 P，采用 **sima1.m** 进行自动分类的典型应用程序，其中取：$r=0.85$，竞争层初始节点取 3 个。下面就是利用 **sima1.m** 程序对 ART1 网络进行训练的程序:

```
% ART1.m
%
P=[1 1 0 1 0 1 1 0;                    % 待分类的输入样本
   0 1 1 0 1 1 1 0;
   1 0 0 0 0 0 0 1;
   1 1 1 0 1 1 1 0;
   0 1 1 1 0 1 1 0;
   1 1 0 1 0 1 1 1;
   0 0 1 0 1 0 1 1;
   0 0 1 0 1 0 0 0;
   1 1 1 0 1 0 0 1;
   0 1 0 1 0 1 1 0;
   1 1 0 0 0 1 1 1;
   0 0 1 0 1 0 0 0;
   0 0 1 0 1 1 0 0];
[R,Q] = size(P);
S = 3;                                  % 选择初始输出节点
W0 = ones(S,R);                         % 网络初始权值
lr = 1;                                 % 选择学习速率
r = 0.85;                               % 选择相似度
disp('Pass 1');
W = sima1(W0,P,lr,r,1);                 % 第一次运行 ART1 网络训练程序
disp('Pass 2');                         % 将运行结果 W 作为初始权值，
W = sima1(W,P,lr,r,1);                  % 进行第二次运行 ART1 网络训练程序
disp('Pass 3');
W = sima1(W,P,lr,r,1);                  % 进行第三次运行 ART1 网络训练程序
W1 = W'
end
```

运行此程序可以看到三次重复训练中每一次对输入样本 P 中每一组矢量的处理结果。

注意在下面的运行结果的显示中，将 ART1 网络的竞争层，也就是 R 层称为第二层:

```
» ART1
Pass 1
Vector 1 resonates Layer-2 neuron 1.
Vector 2 resonates Layer-2 neuron 2.
Vector 3 resonates Layer-2 neuron 3.
Vector 4 resonates Layer-2 neuron 2.
Vector 5 resonates Layer-2 neuron 3.
Vector 6 resets layer-2 neuron 2.
Vector 6 resets layer-2 neuron 1.
Vector 6 resets layer-2 neuron 3.
New Layer-2 neuron created.
Vector 6 resonates Layer-2 neuron 4.
Vector 7 resonates Layer-2 neuron 4.
Vector 8 resonates Layer-2 neuron 1.

Pass 2
Vector 1 resets layer-2 neuron 1.
Vector 1 resets layer-2 neuron 4.
Vector 1 resets layer-2 neuron 2.
Vector 1 resets layer-2 neuron 3.
New Layer-2 neuron created.
Vector 1 resonates Layer-2 neuron 5.
Vector 2 resonates Layer-2 neuron 4.
Vector 3 resonates Layer-2 neuron 3.
Vector 4 resonates Layer-2 neuron 2.
Vector 5 resonates Layer-2 neuron 3.
Vector 6 resonates Layer-2 neuron 4.
Vector 7 resonates Layer-2 neuron 4.
Vector 8 resonates Layer-2 neuron 1.

Pass 3
Vector 1 resonates Layer-2 neuron 5.
Vector 2 resonates Layer-2 neuron 4.
Vector 3 resonates Layer-2 neuron 3.
Vector 4 resonates Layer-2 neuron 2.
Vector 5 resonates Layer-2 neuron 3.
Vector 6 resonates Layer-2 neuron 4.
```

Vector 7 resonates Layer-2 neuron 4.
Vector 8 resonates Layer-2 neuron 1.
Hit any key to see the final weight matrices.

The top-down weights W1:
W1 =

0	1	0	1	1
0	0	1	1	0
1	0	0	0	1
0	0	1	1	1
0	1	0	1	0
1	1	0	1	1
0	0	1	1	0
0	0	1	0	0
1	0	1	0	1
0	1	0	1	0
1	0	0	1	1
0	0	1	0	0
0	0	1	0	0

The top-up weights W2:

W2 =
Columns 1 through 7

0	0	0.5000	0	0	0.5000	0
0.5000	0	0	0	0.5000	0.5000	0
0	0.3780	0	0.3780	0	0	0.3780
0.3536	0.3536	0	0.3536	0.3536	0.3536	0.3536
0.4082	0	0.4082	0.4082	0	0.4082	0

Columns 8 through 13

0	0.5000	0	0.5000	0	0
0	0	0.5000	0	0	0
0.3780	0.3780	0	0	0.3780	0.3780
0	0	0.3536	0.3536	0	0
0	0.4082	0	0.4082	0	0

下面对运行结果记录的 ART1 网络的工作过程进行解释:

在第一轮运行中:

第一个矢量与竞争层中第一个神经元“共振”;
第二个矢量与竞争层中第二个神经元“共振”;
第三个矢量与竞争层中第三个神经元“共振”;
第四个矢量与竞争层中第二个神经元“共振”;
第五个矢量与竞争层中第三个神经元“共振”;
第六个矢量虽然与竞争层中第二个神经元“共振”，但因不满足相似度的要求而被复位搁置。在继续的搜索中，虽然又与竞争层中第一个和第三个神经元分别有了“共振”，但均不满足相似度的要求。此后， ART1 网络自动产生了一个新的竞争层（第四个）神经元作为第六个输入矢量的神经元的代表;
第七个矢量与竞争层中第四个神经元“共振”;
第八个矢量与竞争层中第一个神经元“共振”。

由于有相似神经元的存在，加之新的神经元的出现，使得修正后权矢量在进行第二次运行时，使原先相似的结果又产生了变化:

原先与竞争层中第一个神经元“共振”的第一组矢量在因为虽然“共振”但没有满足相似度要求的情况下，分别复位搁置了竞争层中第一、四、二和三神经元后，又自动产生了一个新的竞争层（第五个）神经元作为其输入矢量的神经元的代表;
第二个矢量不再与竞争层中第二个神经元而是与第四个神经元“共振”;
第三个矢量仍然与竞争层中第三个神经元“共振”;
第四个矢量仍然与竞争层中第二个神经元“共振”;
第五个矢量仍然与竞争层中第三个神经元“共振”;
第六个矢量仍然与竞争层中第四个神经元“共振”;
第七个矢量仍然与竞争层中第四个神经元“共振”;
第八个矢量仍然与竞争层中第一个神经元“共振”。

第三次运行中没有再出现新的变动。

最终的结果是: 8 个输入矢量被分为 5 类，其中，属于第一类的是第八组输入矢量;属于第二类的是第四组输入矢量; 属于第三类的是第三组和第五组输入矢量; 属于第四类的是第二、六和七组输入矢量; 属于第五类的是第一组输入矢量。结果中所给出的权矢量 $W1$ 为五组类型的代表矢量。权矢量 $W2$ 为 $W1$ 归一化处理后的转置矢量。

第七章　面向工具箱的神经网络实际应用

7.1 综　述

近年来，人工神经网络在金融、制造、国防、宇航、医药、通讯、自动控制等领域都有不少成功应用的例子。世界上经营神经网络的软件公司也不断涌现。但是利用神经网络的软件进行应用的设计仍然强调通过有针对性的实验。一个好的神经网络应用的诞生，只有在对网络的种类、结构以及参数经过反复的考虑和修正，对方案通过实验不断地进行改进的基础上而确定。神经网络应用的设计在一定程度上还依赖于经验，所以更加需要重视实验。在掌握了神经网络理论的基础上，加上方便的工具箱的使用，可以使我们在网络的合理结构、网络的优点的利用以及网络性能的提高方面更加集中注意力。下面，我们首先从综合的角度阐述在神经网络设计过程中应考虑的主要因素，为设计神经网络做一些指导性的概括。

7.1.1　神经网络技术的选取

在决定采用神经网络技术之前，应首先考虑是否有必要采用神经网络。对于大多数应用问题，神经网络与最好的计算方法以及统计方法相比并非更加优越。因为对于那些其特征和规律都可求解的问题，可以用逻辑式或数学式精确的描述。尤其在控制系统中，传统的经典控制理论利用线性系统的传递函数，采用根轨迹法、奈魁斯特曲线，以及伯德图等方法，可以系统全面地对线性系统的稳定性等进行分析，设计出满足性能要求的常规比例一微分一积分型控制器。而现代控制理论根据状态空间法求出线性系统的状态空间方程以及转移矩阵，利用可控制性及可观性判据对系统进行分析，通过卡尔曼滤波器、状态观测器等状态反馈手段，设计出专门反馈控制器。另外还可以根据目标函数设计出最优控制器。对于那些能够精确地求出系统的传递函数，或能够用数学表达式来对系统进行描述的系统，则应采用已有的成熟的设计方案来解决问题。

当处理较为复杂的问题，或采用常规方法无法解决或效果不好的问题时，神经网络则能够显示出其优越性。尤其是当对问题内部的规律不甚了解，不能用精确的数学表达式描述其系统，对要求具有容错的任务，如图形的检测与识别或诊断，特征的提取和预测等，神经网络都成为最合适的处理手段。另一方面，神经网络对处理大量数据而又不能用符号或数字方法描述的问题，表现出极大的灵活性与自适应性。

7.1.2 神经网络各种模型的应用范围

神经网络各种模型的应用范围取决于具体的应用任务。下面我们总结一下各种神经网络模型所适用的场所。

（1）函数逼近及图形识别

对于大多数函数逼近、图形识别及信号处理等应用，所选用的网络常常是前馈网络。如果输入矢量的组数为几百个或更多一些，且需要精确的函数逼近值，采用 BP 网络是最恰当不过的了。对于较大的网络，其训练时间过长是一个不足。 BP 网络的另一个不足是网络的训练有可能陷入局部极小值。这些问题可以通过采用改进的 BP 网络的训练方法而得到一定的缓解。

对二进制图形的识别可采用感知器。对连续性数字的图形识别可以采用自适应线性元件。它们的限制均为只能对输入矢量进行线性划分。如果要求网络不仅能够识别图形的类别，而且还能够辨别出大小，则可以采用竞争网络、科荷伦网络或 ART 网络。 ART 网络是集训练与执行过程为一体的网络，可以在不破坏原记忆的情况下学习新的样本，解决了普通竞争网络所存在的遗忘问题。

（2）联想记忆

联想记忆问题通常可分为两类：自联想和异联想。自联想是模式样本自身存储和恢复问题；而异联想则是模式集 X 中的一个模式 x_i，需在另一个模式集 Y 中产生一个对应的模式 y_i。如：原因与结果，声音联想图像，图形联想文字等。无论是自联想，还是异联想问题，它的样本数据均分为两类：数字量和模拟量。有些联想记忆问题是线性映射，而许多联想记忆问题则是非线性映射。

当输入是不完全或缺损的图形，希望通过网络重新恢复完整图形的输出时，可以采用霍普菲尔德网络来解决这类问题。它可以使输入的数据最终收敛到记忆在网络中属于该数据的稳定的平衡点。

（3）数据压缩

数据压缩一般用于数据的传输和存储。有多种网络模型可以用于数据压缩。它们是：

1）用科荷伦网络可以产生 0 、 1 码，对于简单的系统，用此码恢复信息的精度正比于所用码的数目；

2）采用输入与输出层节点数相同的 BP 网络，而使其隐含层的节点数比输入、输出层节点数少得多。训练时，其目标矢量就是输入矢量，这样训练结果使得隐含层中学会了表征输入的特征矢量。将隐含层的输出矢量和输出层的权矢量进行传输储存。当需要解压缩时，可以通过在输出层加上被压缩的隐含层的输出矢量，在其输出端即可得到恢复的原始输入数据；

3）对传网络 CPN 也能够用来进行图像数据的压缩，其中，由科荷伦层执行复杂的变换以便将输入图形的空间维数大大减少，然后，通过格劳斯贝格层的对传映射再恢复原始数据的平均值。

（4）自适应控制

人工神经网络在控制中的应用主要表现在以下几个方面：

1 ）神经网络对于复杂不确定性问题的自适应能力和学习能力，可以被用作控制系统中的补偿环节和自适应环节;

2 ）神经网络对任意非线性关系的描述能力，可以被用于非线性系统的辨识和控制等;

3 ）神经网络的非线性动力学特性所表现的快速优化计算能力，可以被用于复杂控制问题的优化计算等;

4 ）神经网络对大量定性或定量信息的分布式存储能力、并行处理与合成能力，可以被用作复杂控制系统中的信息转换接口，以及对图像、语言等感觉信息的处理和利用;

5 ）神经网络的并行分布式处理结构所带来的容错能力，可以被应用于非结构化过程的控制。

7.1.3 网络设计的基本原则

在实际应用中，面对一个具体的应用领域问题时，通常首先需对领域问题进行分析和抽象，弄清楚究竟要用神经网络方法来求解什么性质类的问题，然后根据问题特点，对各类网络模型进行功能分析，从中初选出某种网络模型，进而对该网络进行特性分析。通过网络系统分析，确定出所选网络是否适合应用问题求解要求，为网络模型改进、组合、创新奠定基础。

网络设计工作与网络系统分析工作密切相关，互为交融。网络设计基本原则主要涉及下述几个方面:

（1）确定信息表达方式

研究领域问题的网络模型表示，将领域问题及其相应的领域知识和经验转化为网络所能表达并能处理的形式。

拿到一个具体的应用领域问题，首先考察其是否能用人工神经网络求解以及是否适宜于用人工神经网络求解。然后必须对其作一番深入的分析，透过现象，抓住本质，抽提问题最本质的主要特征，根据问题的特征和性质进行分类和归纳，将其提炼成适合网络求解所能接受的某种数据形式。例如在对输入样本模式进行分类和识别时，虽然问题的性质各不相同，大致有以下类型:

1 ）已知数据样本;

2 ）已知一些互相间关系不明的数据样本;

3 ）输入／输出模式为连续量;

4 ）输入／输出模式样本为离散量;

5 ）希望把所有输入模式均分类识别开;

6 ）希望识别具有平移、旋转、伸缩等变化的模式。

（2）网络模型选择

包括网络类型的选择，确定激活函数（神经元的 1 ，0 特性）、联接方式（拓扑结构），各计算处理单元间的互相作用方式、神经元的权值和阀值等，其中联接方式包括是全局联接和局域联接。

（3）网络参数选择

确定计算处理单元的数目、输入及输出单元数目、多层网的层数和隐含层单元数目，

以及各个必要的参数的选择。

（4）学习训练算法选择

确定网络学习训练时的学习规则及其改进方法。

在网络设计及其模型选择时，既可采用典型网络模型，也可采用多种网络模型的优点组合方法。另外，还可在典型网络模型的基础上，结合具体应用问题特点，对原网络模型进行变型、扩充等。事实上，在实践中人们往往视具体应用问题，选择合适的模型或依据实际情况发展新的网络模型。

下面将通过具体实际的应用设计来灵活应用 MATLAB 环境下的神经网络工具箱。在具体应用中将侧重于解决问题的思路，实现程序的编写，以及结果的对比。

7.2 神经网络在控制系统中的应用

下面首先介绍非线性控制系统中抵消非线性的基本思想，然后描述一个非线性控制系统问题，并在控制系统中对一个用于函数逼近的神经网络进行定义、设计及其实现。最后进行了仿真实验，并将其实验结果与其他控制方法进行了比较。

7.2.1 反馈线性化

有一些非线性系统，可以很容易地应用反馈线性的方法。所谓的反馈线性化，顾名思义，是利用反馈的控制手段来消除系统中的非线性，以致于使其闭环系统的动力学方程是线性的。反馈线性化的方法可以很容易地实施于可控标准型的控制系统中。当一个系统由下列动力学特性给出:

$$\frac{\mathrm{d}^n x}{\mathrm{d}t^n}=f(X)+b(X)u \tag{7.1}$$

此处 u 是标量控制输出，而 X 是状态变量:

$$X=\left[x\ \frac{\mathrm{d}x}{\mathrm{d}t}\ \cdots\ \frac{\mathrm{d}^{n-1}x}{\mathrm{d}t^{n-1}}\right]^T \tag{7.2}$$

在状态空间表达式中，(7.1)式动力学特性可写为可控制标准型:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}x_1\\x_2\\\vdots\\x_{n-1}\\x_n\end{bmatrix}=\begin{bmatrix}x_2\\x_3\\\vdots\\x_n\\f(X)+b(X)u\end{bmatrix} \tag{7.3}$$

此方程描述了实际系统，我们希望通过加入一个反馈控制项 u，而使其行为能够具有下述理想线性特性:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \\ -k_1x_1 - k_2x_2\Lambda - k_nx_n + Cr \end{bmatrix} \tag{7.4}$$

其中所有 k_i（$i=1,2,\ldots,n$）和 C 均是常数，r 为参考输入。

为了求得反馈控制项 u，令(7.4)式最后一式右端与原系统方程(7.1)右端相等，即可解出控制 u 为:

$$u = \frac{[-K^T X + Cr - f(X)]}{b(X)} \tag{7.5}$$

只要我们采用(7.5)式，对系统进行控制，原系统中的非性线将被消除，被控系统的闭环特性将呈现出(7.4)式所具有的线性特性。并通过选择适当的参数 K 和 C，即可得到期望的任意 n 阶线性系统的响应特性。

7.2.2 问题的提出

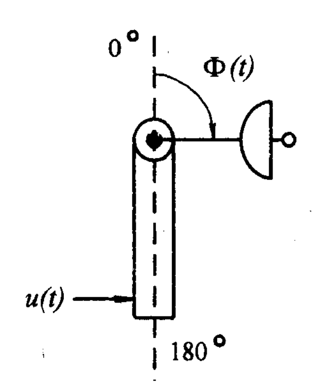


图 7.1 天线被控制系统

考虑一个天线臂仰角控制系统:

这里天线臂角度Φ 是通过改变直流电机中的电流来进行控制的。此系统的方程式可表示为如下非线性系统模型:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ 9.81\sin x_1 - 2x_2 + u \end{bmatrix} \tag{7.6}$$

此处:

$$x_1 = \phi$$

$$x_2 = \frac{d\phi}{dt}$$

其中，u 是送给电机的电流，正电流使天线延顺时针方向转换。$9.81\sin x_1$ 项为其天线臂摆的重力，而$-2x_2$ 项为相对于速度的粘摩擦力。

由以上所给出的被控系统的数学模型，我们可以根据反馈线性化思想来推导出线化控制系统的控制律。首先取:

$$f(X) = 9.81\sin x_1 - 2x_2 \tag{7.7}$$

假定我们期望的闭环系统为下列线性参考模型的动力学表达式给出的响应:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -9x_1 - 6x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 9r \end{bmatrix} \tag{7.8}$$

此处的 r 为期望的输出角位移。

由此可得到完美的对消控制律为:

$$u = 9r - [\,9\ 6\,]X - f(X) \tag{7.9}$$

下面让我们训练一个神经网络来帮助执行这个反馈线性化控制。

7.2.3 神经网络设计

首先假定非性线系统的采样周期为 0.05 秒，那么通过取:

$$\frac{\mathrm{d}X}{\mathrm{d}t} \approx \frac{X(k+1) - X(k)}{\Delta t}$$

可将被控系统的离散时间动力学特性近似地描述为:

$$X(k+1) = X(k) + \Delta t \begin{bmatrix} x_2(k) \\ 9.81\sin x1(k) - 2x_2(k) + u(k) \end{bmatrix}$$

或

$$X(k+1) = F(X(k)) + G(X(k))u(k) \tag{7.10}$$

我们将设计一个神经网络来代替函数 F 以便用它作为控制器 u 中的一部分去抵消系统中的非线性。这个函数为:

$$F(X) = x_2 + \Delta t\,[\,9.81\,\sin x_1 - 2x_2\,] \tag{7.11}$$

假定神经网络对这个函数的逼近是 Nf，那么带有神经网络的非线性消除控制律为:

$$u = \frac{x_2 + \Delta t(9r - [9\quad 6]X) - Nf}{\Delta t} \tag{7.12}$$

(1)网络结构

由(7.11)式可知，所要设计训练的神经网络具有两个输入 x_1 和 x_2。网络设计的目的是能够使其产生逼近函数(7.11)式的非线性输出，所以输出层有一个输出。我们选取一个两层网络的神经模型结构，在隐含层取 10 个神经元，采用双曲正切激活函数；输出层取线性激活函数。网络结构图如图 7.2 所示。

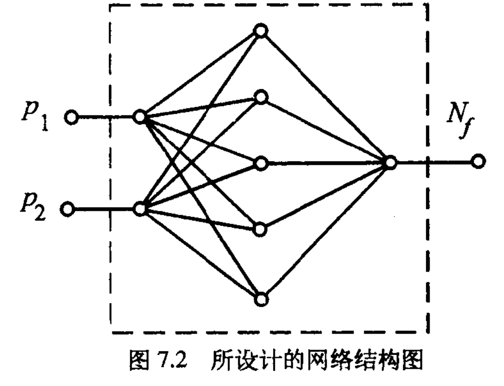


图 7.2 所设计的网络结构图

(2)初始化

为了采用反向传播法进行有监督式的非线性函数逼近，必须给出训练用的输入/输出对。鉴于本例中的非线性函数表达式 $F(X)$已用数学表达式给出，则可以通过随机选取一定数目的工作区的输入，再用 $F(X)$求出目标输出即可。若在实际系统中遇到对其非线性特性不易表达的情况时，可通过对实际被控系统进行输入/输出对的实测记录方式获得数据，再采用适当的控制策略解决问题。本章后面也给出了一个有关用神经网络逼近非线性模型的控制策略。

下面为应用的初始化程序:

```
x1 = [ rands(1:300)*pi, rand(1:100)*pi];
x2 = [ rands(1:300)*pi, zeros(1:100)*pi];
P = [ x1; x2];
dt = 0.05;
```

```
T = x2 +dt*[9.81*sin(x1) - 2*x2);
[R,Q] = size(P);
S1 = 10;
[S2,Q] = size(T);
[W1, B1] = rands(S1,R);
[W2,B2] = rands(S2,S1);
```

用于进行训练的矢量是400个，其中300个是由0到$-\pi$之间的x_1以及$-\pi$到$+\pi$之间的x_2所产生；另外100个是取速度x_2为0，即稳态时，由同样范围内的x_1产生，即我们希望网络在系统稳态时的非线性输出要更加精确以便将其通过反馈控制而完全抵消，使其系统输出达到零稳态误差。

（3）训练

对神经网络的训练采用附加动量法，程序如下：

```
disp_freq = 10;
max_epoch = 5000;
err_goal = 0.02;
lr = 0.0001;
lr_inc = 1.05;  lr_dec = 0.7;
momentum = 0.95;  err_ratio = 1.04;
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_goal];
[W1,B1,W2,B2,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);
```

网络训练在达到误差平方和目标的0.02或达到事先设置的最大循环的次数5 000次后结束。

图7.3和图7.4显示了训练后的神经网络逼近函数F的能力，其中图7.3为在令$x_2=0$, Nf仅作为x_1的函数时的逼近情况；而图7.4为取$x_1=\pi/2$，即将位置固定在水平位置时，Nf对速度响应的情况。由两图可以看出，神经网络能够很好地逼近非线性函数。下面我们将带有神经网络的控制系统的响应特性进行仿真实验，并与其他控制方式的控制效果进行比较。

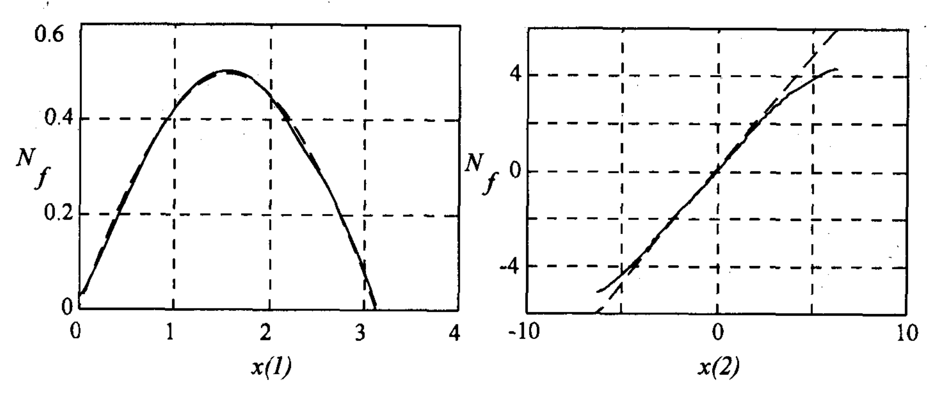


图7.3 $x_2=0$, Nf对x_1的函数逼近　　图7.4 $x_1=\pi/2$，Nf对速度的响应

（4）系统的性能对比实验

现在对神经网络控制系统的性能进行仿真实验，并与其他采用不同控制器的仿真系统的控制效果进行对比。用于对比的控制系统为：

1）具有理想线性参考模型的控制系统；

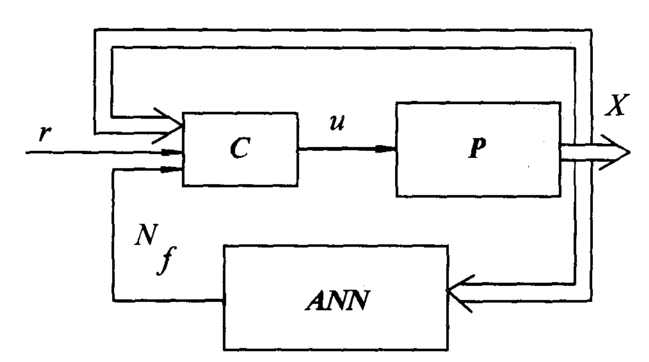
2）具有精确非线性抵消控制器的控制系统；

3）仅采用线性控制器的控制系统。

线性控制器是基于控制系统中被控过程的线性化模型具有理想线性参考模型的响应时设计的。

假定期望天线的定位在水平位置，即$\Phi = 90^{\circ}$。在这种情况下，重心力将驱使天线臂沿顺时针方向转向地面，所以需要一定的电流来保持臂的平衡，这比使天线平衡在0度或180度更加困难。

控制系统结构如图7.5所示，其中反馈控制律由(7.12)式给出，而精确的非线性对消控制器由(7.9)式给出。



C — 控制器；P — 非线性被控过程；ANN — 神经网络非线性逼近器

图7.5　控制系统结构图

线性化的被控过程，在期望平衡点$\Phi = 90^{\circ}$时的线性模型为：

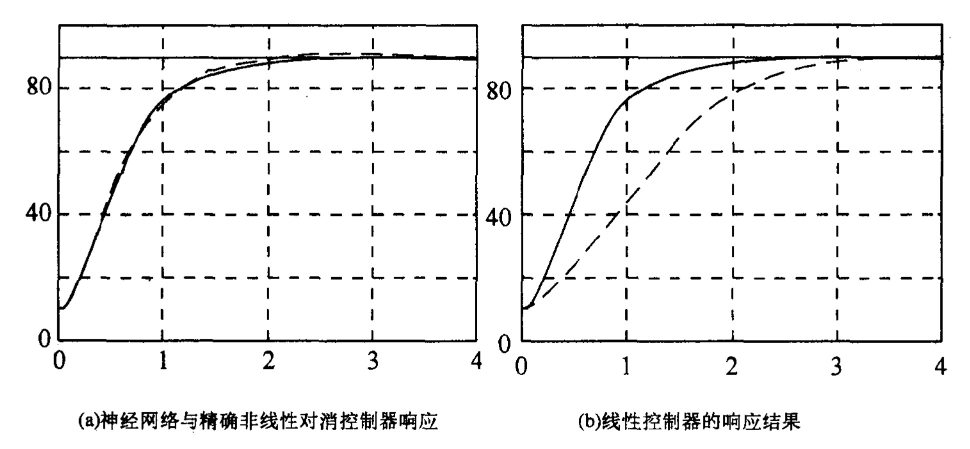
$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -9.81x_1 - 2x_2 + u \end{bmatrix} \tag{7.13}$$

由此线性化控制以及期望闭环参考模型(7.8)式可求得控制律为：

$$u = [-9 \quad -4 \quad 9]\begin{bmatrix} x_1 \\ x_2 \\ r \end{bmatrix} + 10 \tag{7.14}$$

图7.6(a)为神经网络控制器与线性参考模型的响应，以及精确的非线性对消控制器的响应。系统的期望输出为天线臂被保持在夹角$\Phi = 90^{\circ}$，即水平位置上。图中的纵坐标单位为度；横坐标单位为秒。从图7.6(a)中可以看出它们几乎一致。图7.6(b)为非线性系统采用线性控制器的响应结果，其响应速度要慢得多，图中实线为具有理想参考模型系统的响应。图7.7是神经网络控制器、线性控制器和理想对消控制器的响应误差曲线，其中，神经网络系统的响应误差用“o”表示出。虚线为线性控制器的响应误差，实线为理想对消控制

器的响应误差。由此可以看出，神经网络系统的响应是相当出色的。



(a)神经网络与精确非线性对消控制器响应　　(b)线性控制器的响应结果

图 7.6　控制系统的对比响应结果

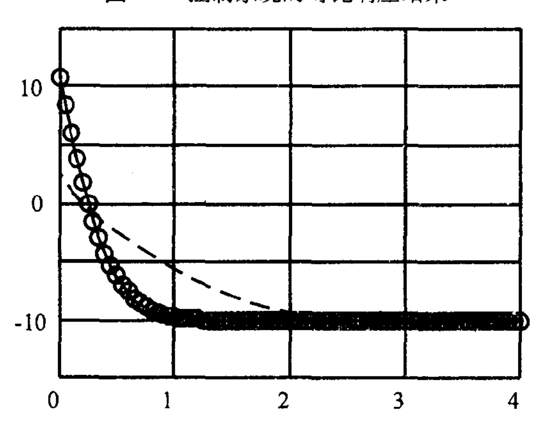


图 7.7　三个控制器的响应误差曲线

7.3　利用神经网络进行字母的模式识别

神经网络可以有助于机器进行模式识别。这是一个非常有实用价值的例子。最典型的用途可以说是我们大家都非常熟悉的辨识信封上的邮政编码。在规定的格子里填上表明地区区号的编码。信封到了邮局后，通过一定的处理程序，就能够自动地用机器而不是人识别其号码，并根据号码内容将其放到合适的位置而达到分类的目的。在这个分信封的过程中，起关键性的数字辨识功能就可以用神经网络来完成。不过神经网络自身仅能够处理数据，所以对于需要神经网络完成的任务，都必须首先将其转化成神经网络所能够接收的输入和输出数据。即使对邮政编码辨识这样一个不复杂的任务，也必须先对信封编码部位进行扫描，并对图像进行预处理，将含有数字的部分用点阵画出其图形，产生一定数目的比如用 0 和 1 数字表示的数字符号的数组后，才能将其送入神经网络的输入端进行辨识。实际上，任何一个实际应用问题，都有一个将其转化成神经网络所能够接受的数据的转化问

题，这种转化往往并不是一件容易的事。

本例则是设计训练一个神经网络能够辨识带有噪声的 26 个英文字母的详细全过程。

7.3.1 问题的阐述

设计训练一个神经网络能够识别 26 个英文字母，意味着每当给训练过的网络一个表示某一字母的输入时，网络能够正确地在输出端指出该字母，那么很显然，该网络记忆住了所有 26 个字母。神经网络的训练应当是有监督地训练出输入端的 26 组分别表示字母 A 到 Z 的数组，能够对应出输出端 1 到 26 的具体的位置。首先必须将每个字母进行数字化处理，以便构造输入样本。考虑用 5 × 7 矩阵的布尔值可以清楚地表示出每个字母，例如字母 A 和 Z 分别可以用 0 、 1 矩阵表示为：

```
letterA = [ 0 0 1 0 0...
            0 1 0 1 0...
            0 1 0 1 0...
            1 0 0 0 1...
            1 1 1 1 1...
            1 0 0 0 1...
            1 0 0 0 1 ]';
letterZ = [ 1 1 1 1 1...
            0 0 0 0 1...
            0 0 0 1 0...
            0 0 1 0 0...
            0 1 0 0 0...
            1 1 1 1 1 ]';
```

每个字母都可以用此 5 × 7 = 35 的元素组成一个字母的列矩阵，那么 26 个字母则分别由表示 26 个字母的输入的列矩阵组成的 26 × 35 的输入矩阵。然后把这 26 个字母送入变量名为 alphabet 中：

```
alphabet = [letterA,letterB,letterC,letterD,letterE,letterF,letterG,letterH,letterI,letterJ,...
            letterK,letterL,letterM,letterN,letterO,letterP,letterQ,letterR,letterS,letterT,...
            letterU,letterV,letterW,letterX,letterY,letterZ];
```

字母的表示如图 7.8 所示。

另一方面，因为目标矢量是希望在每一个字母输入时，在 26 个字母中它所排顺序的位置上输出为 1 ，而在其他位置上的输出为 0 。为此，取目标矩阵为对角线上为 1 的 26 × 26 的单位阵，可用以下 MATLAB 命令来实现：

targets = eye(26);

所有以上关于输入和输出矢量的操作都写进名为 prob.m 文件中备用.

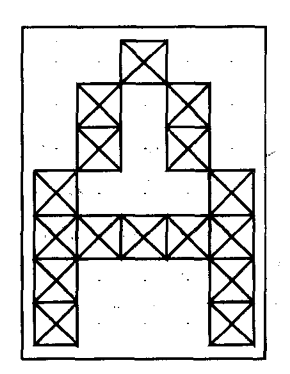
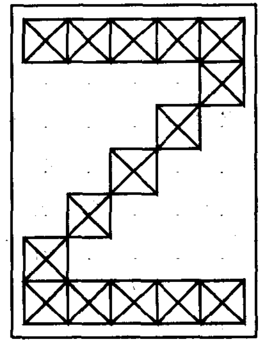



图 7.8 用 5 × 7 矩阵表示的字母 A 和 Z

另外一个很重要的需要考虑的因素是：所要设计的网络应当具有抗干扰能力，即设计出的网络应当能够处理噪声。具有在由一定不规范的输入情况下辨识出正确的字母输入的能力。将干扰噪声进行数字化处理后即变成具有平均值为 0 ~ 0.2 之间变化的随机值。

7.3.2 神经网络的设计

（1）网络结构

所设计的神经网络需要具有 35 个输入节点和 26 个输出神经元。采用输入在(0, 1)范围的对数 S 型激活函数两层 logsig/logsig 网络，这种网络对 0-1 型布尔值是相当完美的。网络取 35-10-26 的结构. 隐含层凭经验取 10 个神经元。如果需要网络有更高的辨识精度，可以再增加一些隐含层神经元数。在本例中，其辨识训练精度对于无噪声的 err_goal 取为 0.1；对于带有噪声的辨识目标误差只要求为 0.6，此目标均在 5 000 次训练内达到。

网络的设计是为了使其输出矢量在正确的位置上输出为 1，而在其他位置上输出为 0。然而，噪声输入矢量可能导致网络的 1 或 0 输出不正确，或出现其他输出值。所以为了使网络具有抗干扰能力，在网络训练后，再将其输出经过一层竞争网络函数 **compet.m** 的处理，使网络的输出只在最接近输入值的输出位置输出为 1，保证在其他位置输出均为 0。

（2）初始化

```
prob.m
P = alphabet;  % alphabet = [ letterA, letterB, ..., letterZ ];
T = targets;   % targets = eye(26);
[R,Q] = size(P);
S1 = 10;
[S2,Q] = size(T);
[W1,B1] = nwlog(S1,R);
[W2,B2] = rands(S2,S1);
```

（3）网络训练

为训练一个网络，使其本身就具有抗噪声的能力，最好的办法是训练一个具有“理想加噪声”输入矢量的识别网络。我们首先训练一个具有较低误差平方和的具有理想输入的网络，以便后面作比较用。然后，用十组“理想加随机噪声”的输入矢量训练神经网络。每一次训练后的权矢量作为下一组输入矢量训练的初始值。训练具有噪声的输入矢量是通过在对两组无噪声矢量训练的同时，加上两对带有随机噪声输入矢量来实现的，这主要是为了保持网络同时具有对理想输入分类的能力。

即使这样，在上述神经网络训练后，也可能出现神经网络虽能够辨识出带有噪声的字母的正确位置，但这却是以牺牲理想无噪声输入辨识的正确性作为代价的，换句话说，此时网络可能出现对理想无噪声输入辨识错误的可能性。为此，应将网络以训练后的权矢量作为初始权值，对无噪声输入再进行训练，以此来保证网络以对理想输入输出的正确性。

上述的所有设计训练思想均在下面的程序中体现出来，其中的网络训练都是用自适应学习速率及附加动量法的函数 **trainbpx.m** 进行的。

1）无噪声字母识别网络的训练程序:

```
disp_freq = 20;
max_epoch = 5000;
err_goal = 0.1;
lr = 0.01;
lr_inc = 1.05;
lr_dec = 0.7;
momentum = 0.95;
err_ratio = 1.04;
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,epochs,TR] = trainbpx(W1,B1, 'logsig ',W2,B2, 'logsig',P,T,TP);
```

训练结束后应当将训练好的网络权矢量存入一个适当数据文件.mat 中，以备网络运行时用，或与后面带有噪声的网络性能作比较时用:

```
save network1.mat W1 B1 W2 B2
```

2）具有噪声的输入矢量识别网络的训练:

为了获得一个对噪声不敏感的网络，我们首先训练一个具有两组理想输入矢量 alphabet 加上两组带有噪声的输入矢量，目标矢量为 4 组期望矢量 targets ，噪声矢量为均值为 0.1 到 0.2 的随机噪声。这迫使网络学习适当地对含有噪声的字母进行正确的识别，不过要求是对理想的输入矢量仍然能够有正确的响应。

对具有噪声的训练，最大训练次数减到 300 且误差目标增加至 0.6 。由于有含有噪声在内的较多输入矢量，其输出误差较高也是正常的。

```
max_epoch = 300;
err_goal = 0.6;
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
    for train = 1:10
        P = [ alphabet, alphabet, ...
            alphabet + randn(R,Q)*0.1), ...
            alphabet + randn(R,Q)*0.2), ...
        T = [ targets, targets, targets, targets, targets ];
        [W1,B1,W2,B2,epochs,TR] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);
    end
```

网络训练完成后，有必要再用理想的无噪声输入矢量对网络再训练一次，以保证网络能够准确无误地识别出理想的字母。最终的训练结果应存入另一个数据文件 network2.mat 中，作为将要被使用的网络权矢量。

（4）系统的性能测试

用神经网络进行模式识别的可靠性，可以通过使用上百个具有随机噪声的输入矢量来测试网络而获得。我们对所设计的神经网络输入任意字母，并在其上加入具有平均值从 0.1 ~ 0.2 的噪声，由此随机产生 100 个输入矢量，通过网络识别后输出，如同前面已经说过的，为了增强网络的抗干扰能力，网络在使用时，其后应再加上一个竞争网络，以使网络对每一个字母的输出只有一个位置为 1，其他元素均为 0。用于性能测试的主要程序如下：

```
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
load network2.mat              % 取训练好的网络权矩阵值
network = [ ];                 % 初始化测试变量
max_test = 100;
noicse_range = 0:0.02:0.2
    for noiselevel = noise_range
    error = 0;
        for i = 1:max_test
        P = alphabet +randn(R,Q)*noiselevel;
        A = logsig(W2*logsig(W1*P,B1),B2);
        AA = compet(A);
        err = err + sum(sum(abs(AA-targets)))/2;
        end
    network = [ network error/max_test/Q ];
    end
plot(noise_range,network*100);
ylabel('Percentage of Recognition Errors'),
xlabel('Noise Level'),
```

由程序可以看出，测试结束后，绘制了辨识的出错百分比。为了对比起见，我们在测试同时加进了对用无噪声的理想字母训练的字母辨识神经网络的测试，并也画出了用其辨识字母的出错率，测试和作图程序与带噪声的上述程序完全相同。两者的比较如图 7.9 所示，图中实线代表用无噪声训练的网络的辨识出错率。虚线为带有噪音训练出的网络情况。由图中可见，两网络都有一定的抗干扰能力，在平均值为 0 ~ 0.1 之间的噪声的影响，都能够几乎 100%地准确无误的辨识出，而在较大的噪声出现后，比如 0.1 ~ 0.2 之间，理想网络明显比带噪声训练网络的出错率要高出许多，在平均噪声为 0.2 的情况下，带噪声的网络的出错率仅为 3.45%，而它的对比结果却达到 4.8%。

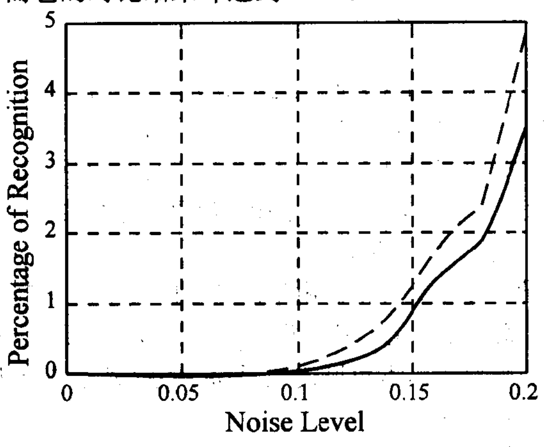


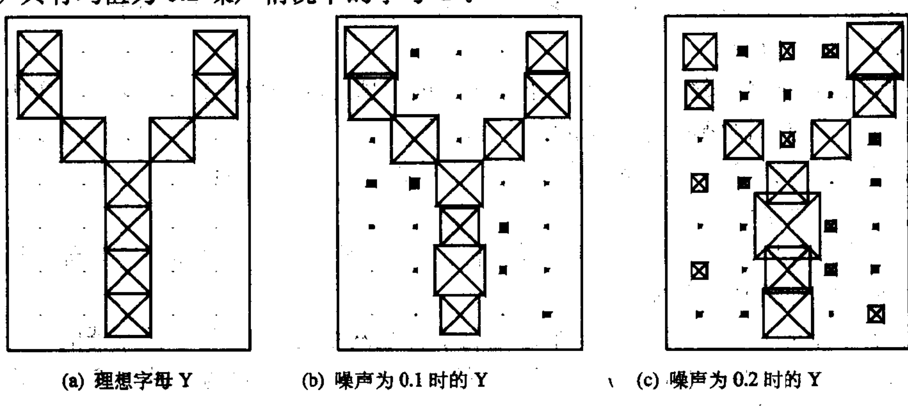
图 7.9　字母辨识的出错率

如果需要更高的辨识精度，可以将网络的训练时间加长，使其训练误差精度更高，或增加网络隐含层中的神经元数。另外也可以增加输入矢量的分辨率，比如采用 10 × 14 点阵。如果需要对较高噪声含量有更好的识别率，也可以在训练时就加大对噪声的含量。图 7.10 给出了三种情况下的字母 Y 的表示:

1) 理想的字母 Y ;

2) 具有均值为 0.1 噪声情况下的字母 Y ;

3) 具有均值为 0.2 噪声情况下的字母 Y 。



(a) 理想字母 Y　　(b) 噪声为 0.1 时的 Y　　(c) 噪声为 0.2 时的 Y

图 7.10　三种情况下的字母 Y

本例表明了一个简单的模式识别系统是如何被设计和实现的。由此可见，其训练过程

不是用几个单一个训练函数就能解决问题的。最后解出的网络性能的好坏，主要还是在于设计人员对问题的认识，设计思想以及运用 MATLAB 工具箱的能力。有了工具箱，可以使我们从繁琐的编程中解放出来，而把精力集中在更好地解决问题和提高网络的性能上。高性能、可靠性以及易实现性才是解决任何问题的最终目标。我们看重的正是工具箱对我们实现目标有极大的帮助，才值得去掌握它。一旦掌握它，则能为我们很好地服务。

7.4 用自组织竞争网络优化模糊神经网络的结构与参数

神经网络与模糊逻辑控制相结合的研究已受到人们的广泛注意，通过神经网络所实现的模糊逻辑控制器的结构具有模糊逻辑推理功能，同时网络的权值也具有明确的模糊逻辑意义。以此方式，用各自的优点弥补对方的不足，可以达到最佳的设计效果。不过，不论采用何种训练方法，所获得的权值，仅仅是模糊神经网络结构在事先固定的情况下的最优，代表模糊标记(即：正小、负大等等)的权值的数目是由设计者在训练之前就已选定，而这个数目为多少经常是根据设计者的经验或是由误差精度的要求来确定。很明显很难选得一个最优值，或称最小值：若此模糊标记的数选得太小，则很难达到期望的目标；若选得太大，网络输出层的模糊控制规则数将以指数形式地增加，这必然造成庞大的网络结构，给权值的训练及网络的实现均带来不便。

本节给出一种解决此问题的设计方法，采用 MATLAB 环境下的神经网络工具箱进行全过程的设计。这是一个较为复杂的综合应用，算得上是一个较为深入的学术研究。其目的是为了使读者有机会接触体会一下作为一项科研应用，应当如何去考虑问题，如何去解决问题，以及在解决问题的过程中的思路、方法和手段等。当然，更为主要的目的是深入了解 MATLAB 环境下的神经网络工具箱的灵活应用。

本应用研究的设计思想是：为了获得同时具有最佳结构和参数的模糊神经网络（Fuzzy-Neural Network,简称 FNN），运用自组织竞争神经网络（Self-organization Competition Neural Network, 简称 SCNN）来优化网络结构。其设计过程分成三步：

1）设计一个具有较多权值的模糊神经网络，其模糊标记数（比如选 7 个），然后，通过训练来优化其权值；

2）选取模糊神经网络中代表模糊隶属函数的权值(包括偏差)作为自组织竞争神经网络的输入矢量。通过该网络的竞争与训练过程，将这些输入矢量的相同的类别自动组合成若干组，竞争后所获得的组合数将成为模糊神经网络的最佳模糊标记数，以此方式将模糊标记的数目减少到最小；

3）重新训练由 2）竞争出的最小模糊标记的 FNN 权值，以获得具有最佳网络结构和最佳参数的模糊神经网络系统。

作为应用研究，下面详细讨论一个经过竞争神经网络处理的 FNN 控制器的设计与训练效果对比的过程。 FNN 控制器所应用的被控对象为一个具有高度非线性摩擦力影响的直流伺服系统。本节除了涉及到神经网络理论与应用外，还涉及到有关模糊逻辑控制以及遗传算法等应用。读者在理解设计原理的基础上，应着重掌握对工具箱的灵活应用之处。

7.4.1 FNN 控制器的设计

首先，设计一个具有 7 个模糊标记，即具有 49 个控制规则的 FNN 控制器，网络结构如图 7.11 所示。可以看出它是一个具有输入层、中间层和输出层的三层神经网络。在功能上，该网络的三层节点是严格对应于模糊逻辑控制的模糊化、规则推理和逆模糊三个步骤的，因而具有明确的模糊逻辑意义。首先，网络输入变量为误差 e 和误差变化 ec；其次，输入层节点的激活函数代表的恰为模糊变量的隶属函数。该层的权值 $W_{ij}^{(1)}$ 和 b_{ij} 的不同也就意味着变化多端的隶属函数的形状和位置。该层的输出 $A^{(1)}$ 代表的就是模糊化的结果——隶属度。再者，中间层是将模糊化得到的隶属度两两相乘的功能。中间层的输出代表着模糊规则的规则强度，将这些强度传递给下一层就可以进行逆模糊了。最后，输出层的各个权值 $W^{(2)}$ 代表了模糊规则，根据重心法的逆模糊方式，只要将它们作为权值与输入，即规则强度加权求和，输出即为模糊控制的输出量。此处采用的模糊推理方式为代数积－加法－重心法。

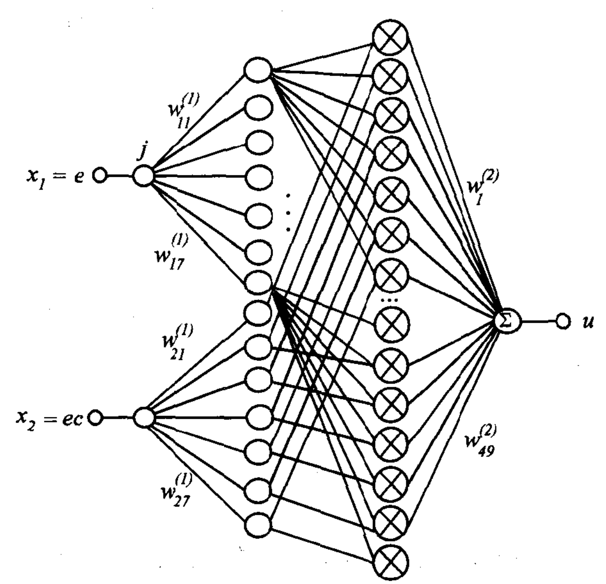


图 7.11 模糊神经网络控制器的结构图

FNN 的输入/输出关系如下:

$$P = [e \quad ec]^T$$

$$a_{ij}^{(1)} = \exp[-(W_{ij}^{(1)} \cdot x_j + b_{ij})^2] \tag{7.15}$$

$$a_k^{(2)} = a_{m1}^{(1)} \cdot a_{n2}^{(1)} = \exp\{-[(W_{m1}^{(1)} \cdot x_1 + b_{m1})^2 + (W_{n2}^{(1)} \cdot x_2 + b_{n2})^2]\} \tag{7.16}$$

$$u = W^{(2)} \cdot A^{(2)} = \sum_{k=1}^{49} a_k^{(2)} \cdot W_k^{(2)} \tag{7.17}$$

其中 $j = 1,2$； $i, m, n = 1,2,\ldots, r1$； $k = 1, 2,\ldots,r1 \times r1$, $r1$ 为模糊标记数。如果脱离具体的神经网络结构，而直接采用乘法－加法－重心法的推理方式和正态隶属函数同样可以得到如上公式。之所以如此，是因为经过这样的从模糊逻辑到神经网络结构的映射和结合，

即可将模糊逻辑的最重要的参数——隶属函数的形状和位置，以及模糊规则转化成了神经网络的权值，这样就可以利用神经网络的自学习、自适应能力来修正和优化它们，也就可以达到优化模糊逻辑控制效果的目的了。

因为在后面的网络设计包括应用中都要多次用到上面的公式运算，而在神经网络工具箱中并没有现成的程序可以调用。所以有必要将其编成子程序以供随时调用。

（1）(7.15)式的编程

我们不妨将由(7.15)式表示的高斯型隶属函数的计算过程写进 gause.m 的 MATLAB 函数中：

```
function a = gause(n,b)                     % 计算高斯型隶属函数
%
if nargin < 1 | nargin > 2                  % 检查输入矢量的个数只能为 1 或 2 个，
   error(' Wrong number of arguments.' );       % 它们分别表示加权输入和 n 和偏差
b,
   end                                      % 不满足输入个数要求时显示出错。

if nargin==1
   z = n;
else
   [nr,nc] = size(n);
   z = n + b*ones(1,nc);
a = exp(-z.*z);                             % 计算高斯型隶属函数
i = find(~finite(a));
a(i) = sign(z(i));
end
```

（2）(7.16)式的编程

(7.16)式是对两输入进行乘法规则的运算，同样也没有现成的程序可用，应当将其编写成一个 MATLAB 函数以备调用： multi.m 。注意输入变量 ecc 是顺序排列的表示隶属度的输出 A^1 的列项量。

```
function   uc = multi(Aeec)            % 函数名与输入、输出变量名 multi.m 、 Aeec 、 uc
%
[ps,pq] = size(Aeec);                  % 求出输入的行和列数
pn = ps/2;                             % 求出两输入变量的个数
pe = [ eye(pn) zeros(pn)]*Aeec;        % 制作两两相乘的矩阵
pec = [ zeros(tempn) eye(pn)]*Aeec;
pec = pec ' ;
 pa = pe(:,1)*pec(1,:);                % 计算矩阵的乘积
```

```
Auc = pa(1,:);                         % 制作 Auc 矩阵
for i = 2:pn
  Auc = [Auc pa(i,:)];
end

for j = 2:pq                           % 制作 Aucj 矩阵
  pa = pe(:,j)*pec(j,:);
  Aucj = pa(1,:);
  for i = 2:pn
    Aucj = [Aucj pa(i,:)];
  end
  Auc = [Auc;Aucj];
end

Auc = Auc′ ;                              % 最后的结果
end
```

最后，(7.17)式可以简单的采用线性函数 purelin.m 获得。

要想对具有图 7.11 所示的模糊推理结构的神经网络进行网络的优化训练，所遇到的困难是：用于训练的期望输入/输出数据是未知的。因为网络的输出就是希望获得的控制量。这个值如果已知的话，那么我们的设计任务就已经完成。为了对网络的参数真正进行优化训练，求出最佳神经网络控制器，可以采用将神经模糊控制器与被控对象串联，并连接成闭环负反馈回路而形成一个控制系统，通过给予闭环系统一个参考输入，以参考输入与闭环系统输出之差的平方为最小作为期望的性能指标来对神经模糊控制器进行参数调整与训练。

为了达到此目的，必须先对被控对象进行模型辨识，然后才能进行控制器参数训练。即整个控制器的设计过程如图 7.12 所示。

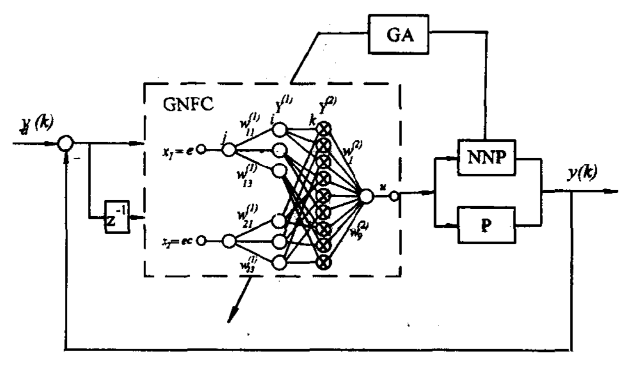


图 7.12　优化系统结构图

为此必须先进行对被控对象模型的网络 NNP 训练以便用于控制器的训练中。

7.4.2 被控对象模型的辨识

被控对象是一个具有严重非线性摩擦力影响的直流伺服电机。它的电流型输入/输出模型结构图如图 7.13 所示。

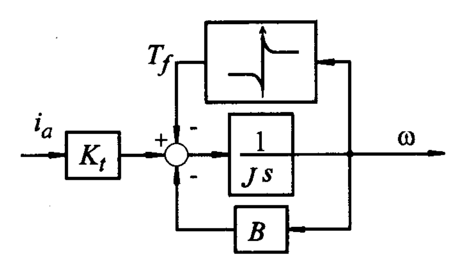


图 7.13 被控直流电机方框图

图 7.14 是实际所测的直流电机的输入/输出特性图。从中可以看出电机中存在着严重非线性摩擦力影响。其非线性摩擦力的模型为图 7.15 所示形状。

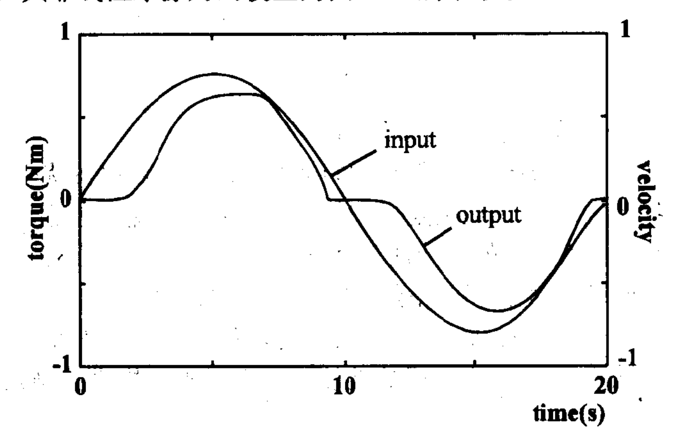


图 7.14 被控过程的输入/输出数据

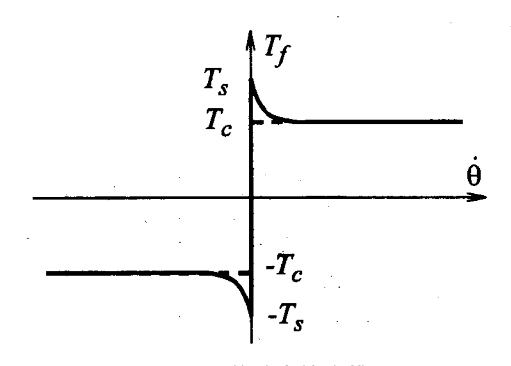


图 7.15 静动摩擦力模型

由于非线性摩擦力的影响，当采用参数辨识方法所获得的线性模型，并以其设计出的常规控制器在对零附近的低速信号或经常变向信号进行跟踪时，总是出现较大的跟踪误差而不能满足高精度的性能要求。为此我们采用人工神经网络进行非线性的模型的输入/输出

特性的建模训练。为了得到动态特性的辨识，特采用带有输出一阶延迟回馈的前向网络。网络内部结构分别采用的是S型和线性激活函数，并通过实验选取五个隐含节点，如图7.16所示。网络有如下的关系：

$$O_i = f\left(W^{(1)} \cdot P + B^{(1)}\right) = f(\Sigma w_{ij}^{(1)} \cdot x_j + b_j^{(1)}), \quad i = 1,2,\ldots,5 \tag{7.18}$$

$$a(k) = f\left(\sum_{i=1}^{5} w_i^{(2)} \cdot o_i + b_i^{(2)}\right) \tag{7.19}$$

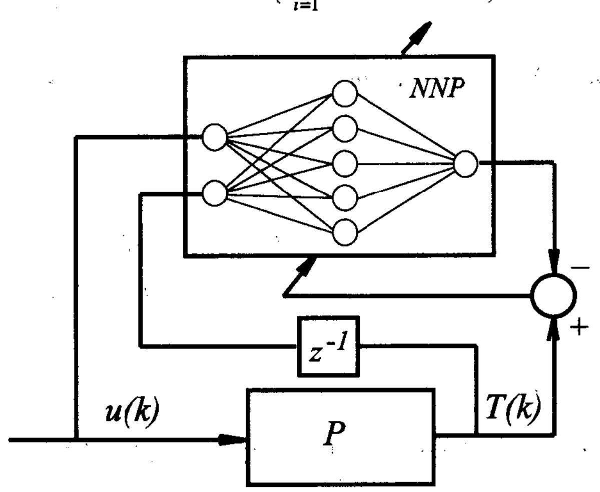


图 7.16　用于辨识的人工网络的结构图

在进行输入/输出特性辨识之前，必须对实际被控对象进行输入/输出数据测试并存入.m文件中作为神经网络函数逼近的输入矢量和目标矢量对。对于用计算机控制的系统首先还要选择采样周期。在本例中所选取的采样周期为0.005秒，采样时间为20秒。这样可获得4000的采样数据，但对于网络训练不能采用如此多的数据，一是没有必要，二是费时，所以在程序中要做适当的处理。本例中将由图7.14中所获得的被控过程的输入／输出数据存入datap.mat中，其中输入变量为in1，输出变量为out1。另外为了能够直接算出物理量（即图7.14中所表出的输入力矩Nm），同时也是为了使样本小于1，在程序中乘了一个根据实际系统获得的比例因子k1 = 2.4015*10^-5(Nm/digit)。下面给出被控过程的输入／输出特性建模的程序，取名为plantpv.m。

```
% plantv.m                               % 文件名
% 被控对象进行输入/输出特性建模
%
clf reset
pausetime = 0.1;
load datapv.mat                          % 装载 in1 和 out1
k1 = 0.000024015;  in = in1* k1;         % in1=17000*sin(0.2*pi*0.005:0.005:20);
out=out1/2^15;                           % 因计算机中采用的是 16 位 D/A 板，...
```

```
%                                                    % 故用 1/2^15 将输出限制在 1 以内。
X11 = in(1:50:4000);                                 % 取 80 个数据对
X21 = out(1:50:4000);
P = [X11′;X21′];                                     % 根据图 7.16 构造训练网络用输入/输出对
T = X21′;
plot(P,T,′+′);                                       % 作训练图
title(′Training Vectors′); xlabel(′Input Vector P′); ylabel(′Target Vector T′); pause

[R,Q] = size(P);
[S2,Q] = size(T);
S1 = 5;                                              % 采用 5 个隐含节点
[W10,B10] = nwtan(S1,R);                             % 采用 N-W 初始条件
W20 = rands(S2,S1)*0.5;
B20 = rands(S2,1)*0.5;

plot(P,purelin(W20*tansig(W10*P,B10),B20));          % 绘制初始权值时的输入/输出特性图
pause2(pausetime);                                   % 设置暂停时间
hold on                                              % 保持图形不被擦除
h = get(gca,′Children′);
h = h(1);

disp_freq = 10;
max_epoch = 2000;
err_goal = 0.001;
lr = 0.01;        lr_inc = 1.05;        lr_dec = 0.7;
err_ratio = 1.04;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec err_ratio];
flops(0);                                 % 采用带自适应速率的反向传播法进行权值训练
[W1,B1,W2,B2,epoch,TR] = trainbpa(W10,B10,′tansig′,W20,B20,′purelin′,P,T,TP);

totalflops = flops;                       % 以下程序显示训练结果并作图
delete(h)
h = plot(P,purelin(W2*tansig(W1*P2,B1),B2),′m′); pause   % 训练结果图
hold off

plotlr(TR(2,:)); pause                    % 学习速率记录曲线
ploterr(TR(1,:)); pause                   % 误差记录曲线
```

```
barerr(A2,T); pause                      %输出误差条形图表
SSE = sumsqr(T-purelin(W2*tansig(W1*P,B1),B2));
fprintf(' \nFINAL NETWORK VALUES:\n ' )
W1
B1
W2
B2
save plant W1 W2 B1 B2                   % 存储训练后的权矢量

fprintf(' Trained for %.0f epochs.\n ' ,epoch)   % 给出结论
fprintf(' Training took %.0f flops.\n ' ,flops);
fprintf(' Average of %.0f flops/epoch.\n ' ,round(totalflops/epoch));
fprintf(' Average of %.0f flops/cycle.\n ' ,round(totalflops/epoch/Q));
fprintf(' Sum squared error goal was %g.\n ' ,err_goal);
fprintf(' Final sum squared error is %g.\n ' ,SSE);
fprintf(' Trained network operates: ' );
if SSE < err_goal
  disp(' Adequately.' )
else
  disp(' Inadequately.' )
  end
```

训练后所得的网络与实际系统的输出曲线如图 7.17 所示。

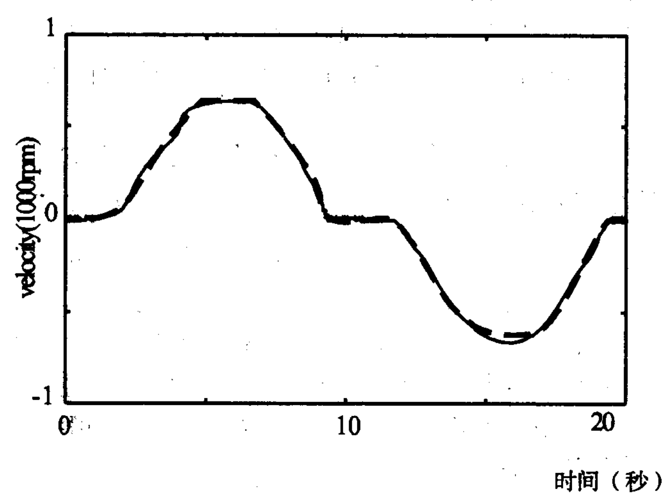


图 7.17 辨识后的网络输出与实际对象输出

7.4.3 FNN 控制器的训练

一旦对被控对象的神经网络进行训练后，则可以按照图 7.12 所示的优化系统结构图对 FNN 控制器进行训练。由于 FNN 不再是一个标准的全联接网络，所以采用 BP 法训练参数的速度相当慢。为了能够加快训练速度，同时避免陷入局部极小值，特采用改进的遗传算法进行参数调整和优化，并用 MATLAB 程序来实现。

首先将所辨识的过程模型与待求的神经网络控制器相串联，并形成负反馈控制回路，再利用改进的遗传算法进行运行仿真，以寻求得到模糊控制中的优化隶属函数和控制规则的组合控制效果。

定义包含所求变量的个体。为了操作方便以及精度的需要，采用实数编码，直接将待处理的权值逐位数字地顺序排列，并转化成数字字符串，形成解的个体。由 N 个个体形成种群。适应度 F_i 采用系统的期望输出与实际系统的输出之差的平方和来定义，即：

$$F_i = \sum_{k=1}^{M} E_i^2(k) = \sum_{k=1}^{M}\left[y_d(k) - y_i(k)\right]^2 \tag{7.20}$$

其中， $i = 1, 2, ..., N$ 为种群中的个体数， k 为个体中待求变量数。优化的目的是使目标函数 F_i 达到某个满意的指标。

在遗传操作的复制过程中，淘汰 0.25N 个低于平均适应度的劣解，并以随机取数的方式补齐，以保持群体交换过程中解的多样性。为了确保搜索的全局最优，在进行交换操作前，首先将本代样本中的最优解直接进化到下一代中。除此之外，每个个体均按一定的比例两两进行多位或一位相互交换，以形成新的个体。复制中新增补的随机个体，尤其在经过数代进化后，与次优个体交换后不断变化出的新个体，可以有效的延缓早熟的出现。当进化接近最优解时，仅由交换操作产生的后代的适应值可能不再比它们的前辈更好，此时将某一位置的自然数变为另一自然数(不同于标准 GA 中改变某一位置上的数字)，目的是为了增加随机性，进而增加多样性。另外，随着进化代数的增加，进行突变的比例也随之增加，以便在接近最优解时的群体中增加更多的新个体。突变比例在整个搜索过程中由 0.05 变化到 0.12 。

下面的例子可以很好的说明交换操作和变异操作的过程。例如：个体中两个变量分别为 0.5712 和–1.2327，当它们以一定方式被选中后，取后三位小数进行交换，形成新的数 0.5327 和–1.2712 ，然后通过变异操作又将其中的某一随机数，如“ 2 ”，即 0.5327 和 –1.2712，变异成“ 3 ”，则形成新一代个体中的变量： 0.5337 和–1.3713，变异操作在越接近最优值、交换操作趋于一致而不再有变化时，越能体现出它的重要性。

在整个系统的训练中，考虑到实际系统模数转换中的饱和特性，也必须在软件训练过程中体现出来，所在程序中设置了一个根据实际值换算出的饱和限定值。采用遗传算法进行 FNN 控制器权值训练的程序如下。

```
%    fnnga7.m                          % 程序名
% 采用遗传算法训练具有 7 个隶属函数的 FNN 控制器权值的程序
%
```

```
clear
pausetime = 0.1;
load plant.mat                        % 取被控对象神经网络的权矢量 W1, W2, B1, B2
k3 = 1.;   k4 = 1.;   k5 = 1.;
fn = 7;   ns = fn*4+fn*fn;            % 由7个模糊标记计算个体中的所含实数编码的总数目ns
T0 = 0.005:0.005:20;
in = 0.6104*k5*sin(0.1*pi*T0);        % 20000/32768 = 0.6104
                                      % 为系统所能跟踪触发的最大幅值
T = in(1:100:4000);                   % 取 40 个数进行训练
[S2,Q] = size(T);                     % 初始化
y = [zeros(1,Q)];    e = zeros([1,Q]); ec = zeros([1,Q]);    P = zeros([2,Q]);
A10 = [zeros(2*fn,Q)]; A11 = [zeros(fn*fn,Q)]; A20 = [zeros(1,Q)]; A21 = [zeros(1,Q)];
SSE = [zeros(1,N)]; WB1 = [zeros(N,ns)*2];
NWB = rands(N,ns)*2;
N = 50;                               % 取种群数

for j = 1:100                         % 开始直至 100 代的遗传进化
  SS = 0;
  for i = 1:N,                        %求 50 组种群
  W10 = [NWB(i,1) 0;NWB(i,2) 0;NWB(i,3) 0;NWB(i,4) 0;NWB(i,5) 0;...
         0 NWB(i,6);0 NWB(i,7);0 NWB(i,8);0 NWB(i,9);0 NWB(i,10);...
         0 NWB(i,11);0 NWB(i,12);0 NWB(i,13);0 NWB(i,14)];
  B10 = [NWB(i,15);NWB(i,16);NWB(i,17);NWB(i,18);NWB(i,19);NWB(i,20);...
         NWB(i,21);NWB(i,22);NWB(i,23);NWB(i,24);NWB(i,25);...
         NWB(i,26);NWB(i,27);NWB(i,28)];
  W20 = [NWB(i,29) NWB(i,30) NWB(i,31) NWB(i,32) NWB(i,33) NWB(i,34)...
         NWB(i,35) NWB(i,36) NWB(i,37) NWB(i,38) NWB(i,39) NWB(i,40)...
         NWB(i,41) NWB(i,42) NWB(i,43) NWB(i,44) NWB(i,45) NWB(i,46)...
         NWB(i,47) NWB(i,48) NWB(i,49) NWB(i,50) NWB(i,51) NWB(i,52)...
         NWB(i,53) NWB(i,54) NWB(i,55) NWB(i,56) NWB(i,57) NWB(i,58)...
         NWB(i,59) NWB(i,60) NWB(i,61) NWB(i,62) NWB(i,63) NWB(i,64)...
         NWB(i,65) NWB(i,66) NWB(i,67) NWB(i,68) NWB(i,69) NWB(i,70)...
         NWB(i,71) NWB(i,72) NWB(i,73) NWB(i,74) NWB(i,75) NWB(i,76)...
         NWB(i,77)];
  B20 = 0;
  CW1 = W10; CB1 = B10; CW2 = W20; CB2 = B20;

% 计算闭环系统误差总和 SSE1
  [SSE1,y,e,ec,cm,A21] = errsqr(Q,T,CW1,CW2,CB1,W1,W2,B1,B2,k3,k4,k5);
```

```
  F(i) = SSE1;                         % 求每一组适应度值
  if F(i) < 0.001, j = j-1,break,end   % 当适应度小于给定的 0.001 ，停止进化
  SS = SS + F(i);                      % 求 50 个种群适应值和
  end

% 复制操作
AF = SS/N;                             % 求 50 个种群适应值的平均值
n = 1;    nn = 1;    min1 = 0;     m = [zeros(1,N)];   mm = 1;   min=1000;
  for i = 1:N,                         % 寻找记录 50 个种群中小于平均适应值 AF 的个体
    dif = AF - F(i);
    if dif >= 0,
    m(n) = i;                          % 记录个数 n 和所处序号 i
    n = n + 1;
    min1 = F(i) - min;                 % 寻找本代最优(最小)适应值
      if min1 <= 0, min = F(i); nn = i; end   % 并把其序号保存到 nn, 其值保存到 min 中
    end
  end

plot(F(1:N)),                          % 绘出本代中 50 个适应值各自的值
pause2(pausetime),
min                                    % 显示本代中最小适应值

  if j == mm*5, nj = j   % 每 5 代显示一次网络输出跟踪目标矢量的情况
  plot([T'   y']), pause2(pausetime), plot([cm'   A21']), pause2(pausetime), mm = mm+1;
  end

% 交换操作
WB1(1,:) = NWB(nn,:);                  % 将本代最优个体放置 1 号位
  for i = 1:n - 1,                  % 将记录中低于平均适应值的个体从 2 号位起顺序排放
  WB1(i+1,:) = NWB(m(i),:);
  end

  if n > 3*N/4, s = 3*N/4;             % 只取少于 s = 3*N/4 的个体数进行淘汰
  else s = n;
  end

  for i = s:N,                         % 用随机值补齐被淘汰的个体
  WB1(i,:) = rands(1,ns)*2;
  end
```

```
  if j <= 30, intwb = fix(WB1*10)/10;  % 30 代前取权值个位及四位小数中的前一位
  else if j >30 & j <= 60,intwb = fix(WB1*100)/100; % 31~60 代取权值小数中的前两位
    else intwb = fix(WB1*1000)/1000;      % 61~100 代取权值小数中的前三位
    end
  end

poiwb = WB1-intwb;                         % 求权值在不同代中的小数部分
NWB(1,:) = WB1(1,:);                       % 将本代最优个体直接进化到下一代

  for i = 2:N/2,               % 将 2 到 N/2 与 N-1 到 N/2 - 1 的个体之间的权值的小数...
  NWB(i,:) = intwb(i,:) + poiwb(N+1-i,:);   % 进行交换
  end
  for i = 1 + N/2:N,
  NWB(i,:) = intwb(i-N/2,:)+poiwb(i,:);
  end

% 变异操作
  if j <= 40, jk = N/2;                          % 选择变异的频率
  else if j > 40 & j <= 70, jk = fix(N/3);       % 从每 25 个个体中变异一次增长到
    else if j > 70 & j <= 100, jk = fix(N/4);    % 每 18 到 12 个个体中变异一次
    end
  end

  for i = 5:jk:N,
  k = fix(rand*10);                              % 随机取一个 9 以内的整数
  mu1 = num2str(NWB(i,:));                       % 将所选定的个体中的权值数码中与随机
  mu2 = strrep(mu1,' k ',' k-1 ');                            % 数 k 相同的数变为 k - 1
  hh = sscanf(mu2,' %f ')' ;
  [in,im] = size(hh);
    if im < 77, NWB(i,:) = [hh 0.];
    else NWB(i,:) = hh;end
    end
  end
end

wbg7 = NWB(1,:);                           % 取最佳权值
save fnnwb7 wbg7                           % 将训练结果存入文件 fnnwb7 中
```

从上面的程序中可以看到，调用了另外一个用来计算闭环系统误差总和 SSE1 的子程序： errsqr.m 。这个程序实际上是计算了闭环系统的前向回路的输出，即从误差到系统输出的计算过程，是体现网络性能优劣的关键所在，是系统运行的核心。其程序清单如下：

```
function [SSE1,y,e,ec,cm,A21] = errsqr(Q,T,W1,W2,B1,PW10,PW20,PB10,PB20,k3,k4,k5)

e(1) = rand*.01;     ec(1) = rand*.01;   P = zeros([2,Q]);

for k = 1:Q,
  P(:,k) = [e(k);ec(k)];                    % 构造输入矢量
  A10(:,k) = ovensig(W1*P(:,k),B1);         % 计算 FNN 神经网络的输出 A20
  A11(:,k) = multi(A10(:,k));
  A20(k) = purelin(W2*A11(:,k));

  A21(k) = A20(k)*k4;                       % 检查所计算出的控制量是否正向饱和
    if A21(k) > 0.7869
    cm(k) = 0.7869;              % 32768*0.000024015 = 0.7869
    elseif A21(k) < -0.7869                  % 检查所计算出的控制量是否负向饱和
      cm(k) = -0.7869;                      % 饱和时输出最大饱和值
    else
      cm(k) = A21(k);                       % 未饱和则输出计算值 A21
    end

xp(:,k)=[cm(k);T(k)];                       % 构造被控对象网络输入矢量
y(k) = k5*purelin(PW20*tansig(PW10*xp(:,k),PB10),PB20);   % 计算系统输出值
E(k) = T(k) - y(k);                         % 计算系统输出误差值
ec(k+1) = e(k);                                 % 计算系统下一时刻的误差值
e(k+1) = E(k)*k3;
end

SSE1 = sumsqr(E);                           % 计算系统的误差平方和
```

训练结束后可以得到优化后的 FNN 权值。这是一个具有 77 个参数的人工神经网络，是一个较为庞大的网络。此时的权值优化严格地说是在模糊标记数固定为 7 的情况下的权值优化，而并不是最简网络结构下的优化值。这意味着在相同的优化指标下，有可能用更加简单的网络来实现控制器的设计。我们知道 FNN 的权值数目是由设计者事先选定的模糊标记数来确定的。那么如何求得这个最少模糊标记数呢？下面利用竞争网络来实现这一目的。

7.4.4 采用 SCNN 优化模糊标记数与性能对比

对于上述具有 7 个模糊标记的模糊神经网络，在总共 77 个参数中，有 28 个权值和偏差代表不同输入的隶属函数。它们可以被分为两组：一组由误差变量 e 的 7 个权值和 7 个偏差组成，另一组包含误差的变化变量 ec 的 7 个权值和 7 个偏差值，现在将这两组数据，作为 SCNN 的输入矢量 P。竞争层的节点数选为 5，最大循环数 $N = 500$，相似度偏差 $b =$ -0.9。在对输入矢量归一化处理后，开始进行竞争和权值的训练，训练目标是使相似的输入矢量聚成同一类型以达到减少输入矢量数组的目的。所有的设计与训练程序均由 MATLAB 及其中的神经网络工具包完成。其中由 28 个代表不同输入的隶属函数的权值和偏差转化为 SCNN 的输入矢量 P 的程序为 peec7.m:

```
% peec7.m                          % 文件名
% 将 FNN 网络权值 W1 和 B1 转变成变量 PE 和 PEC
%
load fnnwb7
NWB = wbg7;
PE = [NWB(1,1) NWB(1,2) NWB(1,3) NWB(1,4) NWB(1,5) NWB(1,6) NWB(1,7);
   NWB(1,15) NWB(1,16) NWB(1,17) NWB(1,18) NWB(1,19) NWB(1,20) NWB(1,21)];
PEC = [NWB(1,8) NWB(1,9) NWB(1,10) NWB(1,11) NWB(1,12) NWB(1,13) NWB(1,14);
  NWB(1,22) NWB(1,23) NWB(1,24) NWB(1,25) NWB(1,26) NWB(1,27) NWB(1,28)];
end
```

而进行竞争分类的程序必须对误差 e 与误差的变化 ec 分别实施，下面给出对变量误差进行竞争分类的程序 compet7.m：

```
% compet7.m                        %文件名
%
clf reset
pausetime = 0.05;
peec7
P = normc(PE);
[R,Q] = size(P); S = 5;
% W0 = randnr(S,R);                           % 也可以取随机初始值
W0 = [-0.7591     0.6449                      % 一组较好的初始值
        0.9988    -0.0482
        0.7080     0.6982
       -0.7067    -0.6585
       -0.3425     0.9340];
```

```
plot(cos(0:.1:2*pi),sin(0:.1:2*pi),'--')                % 作输入矢量图
axis('equal')
hold on
plotv(P);
h = plot(W0(:,1)',W0(:,2)','.');                        % 作初始权矢量图
title('Input Vectors (lines) & Weight Vectors (+)');
xlabel('P(1,q)    W(i,1)');
ylabel('P(2,q)    W(i,2)');

disp_freq = 10;     max_cycle = 500;   lr = 0.05;

% 在不作图的情况下可以用命令: TP = [disp_freq max_cycle lr];
% 以及 W = trainc(W0,P,TP); 来完成下列竞争过程

W = W0;
flops(0)
LW = W;
for cycle=1:max_cycle

  % PRESENTATION PHASE
  q = fix(rand*Q) + 1;
  A = compet(W*P(:,q));                    % 竞争
  i = find(A == 1);                        % 利用科荷伦规则进行权值调整
  dW = learnk(W,P(:,q),i,lr);
  W = W + dW;
  temp = flops;                            % 显示训练过程
  if rem(cycle,disp_freq) == 0
    delete(h)
    h = plot(W(:,1)',W(:,2)','+');
    LW = W;
    pause2(pausetime);
    end
  flops(temp);
  end

totalflops = flops;
pause
hold off
```

```
fprintf(' \nFINAL NETWORK VALUES:\n ');        % 总结最后结果
W
fprintf('  Trained for %.0f cycles.\n ' ,max_cycle)
fprintf('  Training took %.0f flops.\n ' ,totalflops);
fprintf('  Average of %.0f flops/cycle.\n ' ,round(totalflops/max_cycle));

YE = hardlim(W*P-.90)                           % 检验输出结果
```

对于变量误差变化的竞争，只要把上面程序中的变量 PE 改为 PEC，重新运行一遍程序即可获得结果。

图 7.18 给出了竞争和训练之后的输入与 SCNN 权值矢量的分布图。其中“线”表示输入矢量，“+”代表权值矢量值，因为参数被归一化处理，所以输入矢量具有单位模数，所有的输入矢量值均落在单位圆周上。从图 7.18 可以看出，4 种权值代替了 7 输入矢量，另外，5 个权值矢量中还多余出一个。

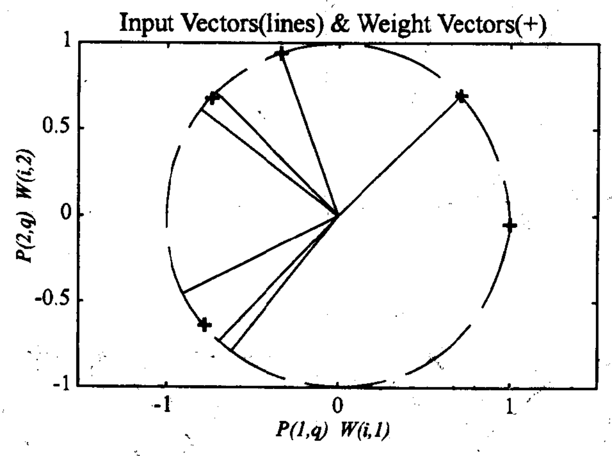
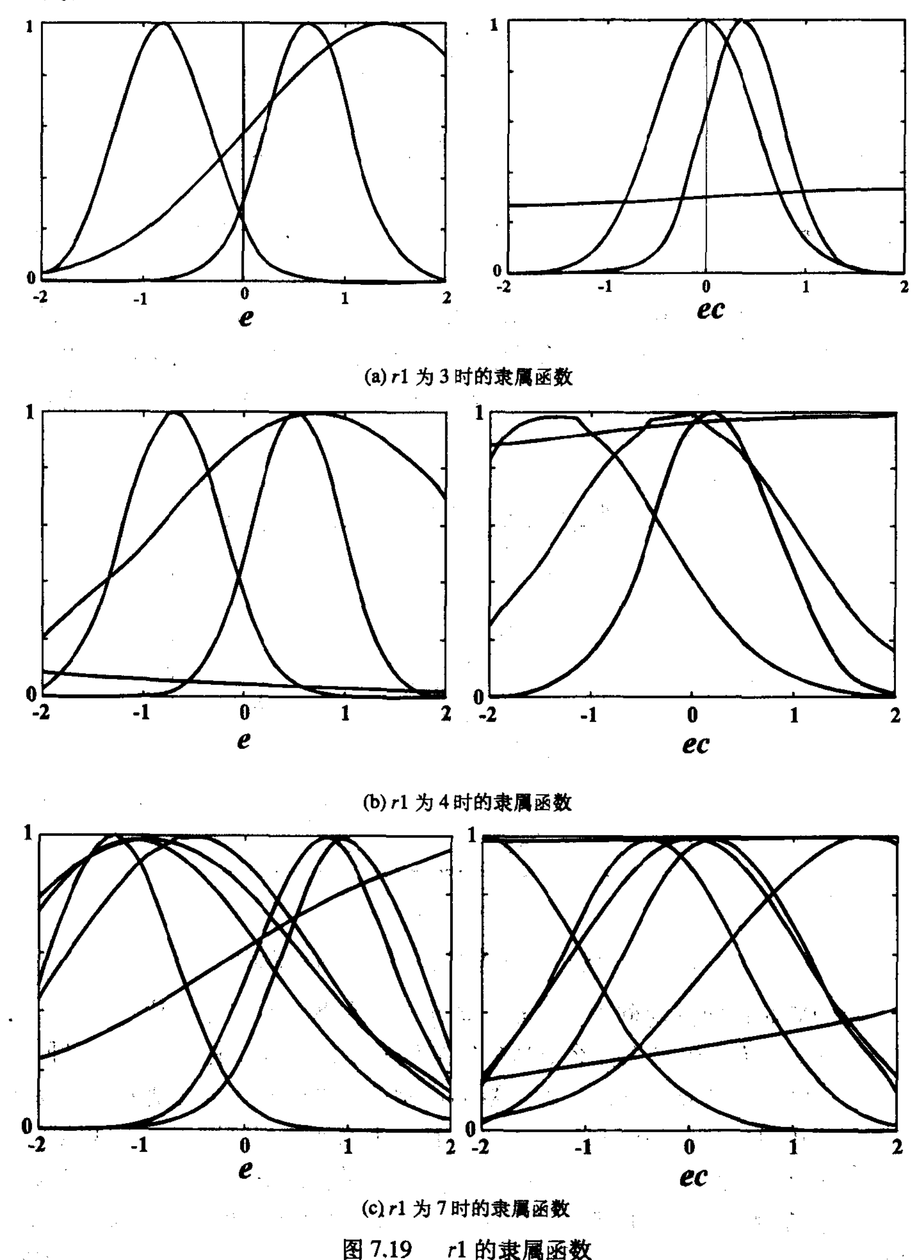


图 7.18　输入矢量(线)和权值矢量(+)的分布图

按照一般的概念，通常模糊标记的数目的选择是单数，且最小数应当取 3：负、零和正，而此处在经过竞争训练后为什么对所研究的问题给出了 4 个标记数？为了回答这个问题，同时为了比较，对具有 3 个模糊标记数的模糊神经网络控制器也进行了训练设计，同样采用改进的遗传算法优化网络权值。不幸的是，不论如何训练网络，不能达到期望的误差值，控制系统也不能跟踪输入信号，图 7.19(a)给出了训练后所得到的一种情况：能够很好地跟踪正向参考输入。而负向跟踪产生较大误差时的隶属函数曲线，图 7.19(b)和(c)分别为模糊标记为 4 和 7 时的隶属函数曲线。比较三者可以看出，不论选取几个模糊标记，总存在一条平坦的隶属函数曲线，此隶属函数是用来克服被控过程中存在的非线性摩擦力。当模糊标记数为 3 时，除了一个用于非线性补偿外，只剩下两个可以用来控制动力学特性，很明显是不够的，从图 7.19(a)中可以看出，*ec* 中只有一个正和零的隶属函数，然而没有表示负的隶属函数，所以系统不能很好地跟踪负向信号，而当模糊标记数为 4 时，如图 7.19(b)所示，一个用来消除非性摩擦力，三个用来分别跟踪负、零和正向信号，以达

到最佳控制效果，当模糊标记数多于 4，那么多余的隶属函数或出现在 3 个主要的隶属函数附近，或协助克服非线性，所有这些多余的“曲线”，均能通过 SCNN 的训练而被 4 条隶属函数代替。



(a) $r1$ 为 3 时的隶属函数

(b) $r1$ 为 4 时的隶属函数

(c) $r1$ 为 7 时的隶属函数

图 7.19　$r1$ 的隶属函数

在通过 SCNN 的竞争和训练后，则获得最小的模糊标记的数目，然后，用 4 个模糊标记重新构成一个新的模糊神经网络，并可再次应用上述同样的过程训练出网络权值。

图 7.20 给出速度跟踪系统在模糊神经网络控制器作用下，对梯形输入信号的响应，其中，图 7.20(a)则通过 SCNN 训练结果获得的具有 4 个以及原有 7 个模糊标记的模糊神经网络控制器的响应信号，纵轴为速度，单位为 1 000 转/分钟；图 7.20(b)为其响应误差信号，

纵轴的误差单位为转/分钟，横轴单位均为时间。图中实线为具有 4 个模糊标记的 FNN 控制器的误差。从图中可以看出，两个控制系统具有完全相同的跟踪输入信号能力，这意味着：前者可以取代后者，同时也是最小数目和最简结构，因为当其数目小于 4 时，模糊神经网络控制器不能很好地工作。所以可以说，由通过 SCNN 的训练所获得的模糊标记数所组成的模糊神经网络结构及其训练出的权值具有最优结构与权值。

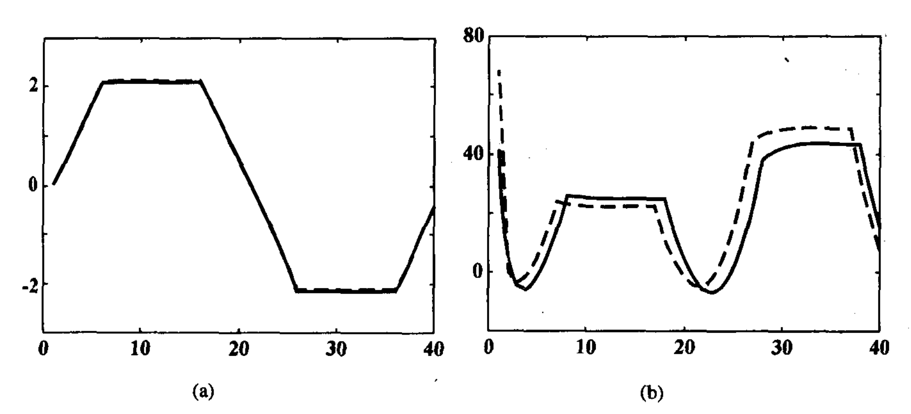


图 7.20　输入信号与不同情况下的控制系统响应(a)与误差信号(b)

下面只给出绘制模糊标记为 4 时的图 7.19(b)中的隶属函数的程序。

```
% msf4.m     (fig7.19b)              文件名
% 绘制具有 4 个模糊标记的隶属函数图形
%
load fnnwb4                    % 取 FNN4 的权矢量
T0 = 0.005:0.005:20;
in = 0.6104*sin(0.1*pi*T0);  % 20000/32768 = 0.6104  给出系统的最大跟踪量
T = in(1:100:4000);
e = -2:0.05:2;  [n,N] = size(e);     ec = e;
P = [e;ec];
NWB = fwb4;
i = 1;
W1= [NWB(i,1) 0;NWB(i,2) 0;NWB(i,3) 0;NWB(i,4) 0;NWB(i,5) 0;...
        0 NWB(i,6);0 NWB(i,7);0 NWB(i,8)];
B1 = [NWB(i,9);NWB(i,10);NWB(i,11);NWB(i,12);NWB(i,13);NWB(i,14);...
          NWB(i,15);NWB(i,16)];
W2 = [NWB(i,17) NWB(i,18) NWB(i,19) NWB(i,20) NWB(i,21)...
         NWB(i,22) NWB(i,23) NWB(i,24) NWB(i,25)...
         NWB(i,26) NWB(i,27) NWB(i,28) NWB(i,29) NWB(i,30)...
         NWB(i,31) NWB(i,32)];
```

```
lfun = ovensig(W1*P,B1);          % 计算隶属函数
plot(e,lfun(1,:)),                % 绘制误差隶属函数图
hold on
plot(e,lfun(2,:)),
plot(e,lfun(3,:)),
plot(e,lfun(4,:)),
hold off,pause,

plot(ec,lfun(5,:)),               % 绘制误差的变化隶属函数图
hold on
plot(ec,lfun(6,:)),
plot(ec,lfun(7,:)),
plot(ec,lfun(8,:)),
hold
end
```

进一步的系统性能比较可以通过对相同的被控过程使用常规控制方案来加以对比。图7.21对给出了被控过程采用极点配置法所获得的PI控制器，在相同输入信号作用下的误差响应，其中，纵轴为误差信号，单位为转/分钟。从中可以清楚地看出：虽然PI控制器对常数干扰不产生误差，但被控过程中所存在的零附近的非线性摩擦力的影响，对系统的响应产生了最大值约每分钟180转的误差，而从其对比结果图7.20(b)中可以看出：其最大误差只有约每分钟40转。由此可以显示出FNN控制器的优于常规控制器之处。当然不仅如此，我们的研究还达到了FNN控制器获得了结构与参数的双重最优的目的。

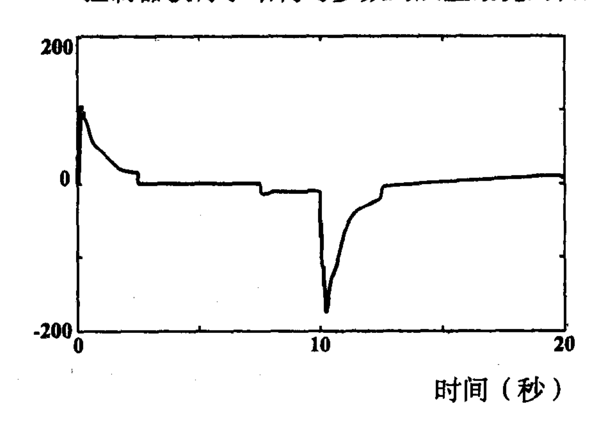


图7.21　常规控制器对图7.20(a)输入信号的响应误差

附 录 MATLAB 神经网络工具箱函数一览表

分析函数

函数名	意 义
errsurf	计算误差表面
maxlinlr	计算最大学习速率
presflop	计算浮点运算的表达式

δ 函数

函数名	意 义
deltalin	对 pureline 的 δ 函数
deltalog	对 logsig 的 δ 函数
deltatan	对 tansig 的 δ 函数
getdelta	对给定函数的 δ 函数

设计函数

函数名	意 义
solvehop	反馈网络设计
solvelin	线性网络设计

初始化函数

函数名	意 义
nwlog	对具有对数 S 型激活函数的神经元产生 N-W 随机数
nwtan	对具有正切 S 型激活函数的神经元产生 N-W 随机数
randnc	产生归一化列随机数
randnr	产生归一化行随机数
rands	产生对称随机数

矩阵

函数名	意　义
normc	计算归一化列矩阵
normr	计算归一化行矩阵
pnormc	计算伪归一化列矩阵
sumsqr	计算平方和

学习规则

函数名	意　义
learnbp	反向传播学习规则
learnbpm	带有附加动量法的反向传播学习规则
learnh	海布学习规则
learnhd	带有衰减因子的海布学习规则
learnis	内星学习规则
learnos	外星学习规则
learnk	科荷伦学习规则
learnp	感知器学习规则
learnwh	W - F学习规则

邻层函数

函数名	意　义
neighb1d	一维邻层函数
neighb2d	二维邻层函数

激活函数

函数名	意　义
compet	竞争层激活函数
hardlim	硬限制激活函数
hardlims	对称硬限制激活函数
logsig	对数S型激活函数
purelin	线性激活函数
satlin	饱和线性激活函数
tansig	正切S型激活函数

绘图

函数名	意　义
barerr	每个输出矢量的条形图表
hintonw	绘制具有平方的权值图
hintonwb	绘制具有平方的权值和偏差的图
ploterr	绘制网络误差与训练次数的关系图
plotfa	绘制目标模式与网络函数的逼近图
plotlr	绘制网络学习速率与训练次数的关系图
plotmap	绘制具有任意邻近函数的特性图
plotpc	绘制感知器对硬限制神经元的分类图
plotpv	绘制感知器对硬限制神经元的训练矢量图
plottr	绘制网络误差记录以及自适应学习速率
plotv	绘制始于坐标原点的单位模长矢量线

仿真

函数名	意　义
simhop	霍普菲尔德网络仿真

网络训练

函数名	意　义
trainbp	采用反向传播法的训练
trainbpa	带有附加动量法的反向传播训练
trainbpm	带有带有自适应学习速率的反向传播训练
trainbpx	带有附加动量法和自适应的反向传播训练
trainc	训练竞争层
trainfm	训练特性图
trainp	采用感知器规则的训练
trainwh	采用 W-H 规则的训练

参考文献

[1] McCulloch W. C. and Pitts W., A logical Calculations of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, Vol. 5: 115-133, 1943

[2] Rosenblatt F., The Perceptron: a Probabilistied Model for Information Storage and Organization in TL Brain, Psychological Review Vol. 65: 386-408, 1958

[3] Minkey M. and Papert S., Perceptrons, MIT press, 1969

[4] Widrow B. and Hoff E., Adaptive Switching Circuits, IRE WESCON Convention Record, Part 4, Computers, Man-Machine Systems: 96-104, 1960

[5] Hebb D. O., The Organization of Behavior, John Wiley, New York, 1949

[6] Grossberg S. A., Studied of Mind and Brain, Reidel Press, Dordrecht, Holland, 1982

[7] Kohonen T., Correlation Matrix Memories, IEEE Trans. on Computers, Vol. 21: 352_359, 1972

[8] Hecht-Nielsen R., Theory of the Back Propagation Neural Network, Proc. of IJCNN, Vol. 1: 593-603, 1989

[9] Rumelhart D. E., Hinton G. E., Williams R. J., Learning Internal Representations by Error Propagation, Rumelhart D. and McClelland J. editors, Parallel Data Processing, Vol. 1, Chapter 8, the M.I.T. Press, Cambridge: 318-362, 1989

[10] Patrick K. Simson, Artificial Neural Systems - Foundations, Paradigms, Applications, and Implementations, Pergamon Press, 1990

[11] Li J. H., Michel A. N. and Porod W., Analysis and Synthesis of a class of neural Networks: Linear Systems Operating on a Closed Hypercube, IEEE Trans. on Circuits and Systems, Vol. 36, No. 11: 1405-1422, Nov. 1989

[12] Neural Network Toolbox User's Guide, The Math Works Inc, 1993

[13] Nguyen D. H. and Widrow B., Neural Networks for Self-Learning Control Systems, IEEE Control Systems Magazine, April 1990: 18-23

[14] Widrow B. and Bilello M., Nonlinear Adaptive Signal Processing for Inverse Control, World Conference on Neural Networks' 94: III-3-13, 1994

[15] Feldman J. A., Connections Models and Their Applications: Introduction, Cognitive Science, Vol. 9: 1-2, 1985

[16] Cong S., De Carli A, A Compound Optimized Control Strategy, 5th Symposium on Low Cost Automation, Sept. 8-10, Shenyang, 1998

[17] Cong S., Wu G, Li G. D., The Decrease of Fuzzy Label Number Using Self-organization Competition Network, International ICSC?IFAC Symposium on Neural Computation NC'98, Sept. 23-25, 1998, Vienna, Austria

[18] De Carli A. and Cong S., Intelligent Neural Network Controller for a Position Control System, IFAC Workshop Motion Control, Munich, Germany, Oct. 9-11, 1995: 189-196

[19] 张立明.人工神经网络的模型及其应用，复旦大学出版社， 1993

[20] 李孝安，张晓渍.神经网络与神经计算机导论，西北工业大学出版社， 1994

[21] 薛定宇.控制系统计算机辅助设计— MATLAB 语言及应用，清化大学出版社，1996
[22] 胡上序，程翼宇.人工神经元计算导论，科学出版社，1994
[23] 黄德双.神经网络模式识别系统理论，电子工业出版社，1996
[24] 王俊普.智能控制，中国科技大学出版社，1997
[25] 陈燕庆，鹿浩.神经网络理论及其在控制工程中的应用，西北工业大学出版社，1991
[26] 丛 爽.神经网络在电机非线性补偿中的设计与实现,《中国控制会论文集》，833-837，1996
[27] 丛爽，钱镇.用于电机中非线性补偿的变参数双模糊控制器，《微特电机》，1997, 25(3):2-4
[28] 丛爽.运动控制中模糊逻辑控制的应用，电气自动化，1997, 6 : 16-18
[29] 丛爽.运动控制中先进控制策略的研究综述，《微特电机》, 1998, 26(1) : 2-8
[30] 杨开宇，陆闽宁.模糊控制算法的研究. 东南大学学报. 1995, 25(5a): 254-258
[31] 应行仁，曾南.采用 BP 神经网络记忆模糊规则的控制. 自动化学报. 1991,17(1):63-67
[32] 张乃饶.用遗传算法优化模糊器的隶属函数. 电气自动化. 1996,1: 4-6
[33] Goldberg D E. Genetic Algorithms in Search, Optimization and machine Learning. Mass: Addison-Wesley, 1989
[34] Morimoto, T. and Hashimoto,Y.(1996). Fuzzy Control for Fruit Storage Optimization Using Neural Networks and Genetic Algorithms, Proceedings of IFAC 13th Triennial World Congress, 375 - 380
[35] Dayhoff J.E.,Neural Network Architectures - An Introduction, Van Nostrand Reunhold, 1990
[36] Brown, M., and Harris, C. J., Neurofuzzy Adaptive Modeling and Control, Hemel Hempstead, V. K.: Prentice Hall, 1997
[37] Hopfield J. J. and Tank D. W., Neural Computation of Decisions in Optimization Problem, Biological Cybernetics, 1985, Vol. 52: 141-152
[38] 方建安，邵世煌.采用遗传算法学习的神经网络控制器，控制与决策，1993，8（3）: 208 — 212
[39] 席裕庚，柴天右，恽为民.遗传算法综述，控制理论与应用，1996，13（6）: 687 — 708
[40] 晏建军，何水保.利用遗传算法简化神经网络结构，计算机工程，1995，21（5）: 56 — 61
[41] 张峻，陈坚，席裕庚.MATLAB 软件包与控制系统设计及仿真，电气自动化，1997，4：7 — 8
[42] 许力.一种局部化的反向传播网络，控制与决策，1995，10（2）：148 — 152
[43] 刘卫国，李钟明，吴斌，马瑞卿.神经网络在电机控制系统中的应用，微特电机，1996，4：12 — 14
[44] 袁震东.自动控制学科面临的挑战与机遇，电气自动化，1995，1：42 — 43
[45] 方剑，席裕庚.神经网络结构设计的准则和方法，信息与控制，1996，25（3）: 156 — 164
[46] Antsaklis P. J., Neural Networks in Control Systems, IEEE Control Systems Magazine, April 1990: 3-5

[47] Naomi E. L. and William S. L., Using MATLAB To Analyze and Design Control Systems, The Benjamin/Cummings Publishing Company, Inc., 1992
[48] Frederick D. K., chow J. H., Feedback Control Problems - Using MATLAB and the Control System Toolbox, PWS Publishing Company, 1995
[49] 焦李成.神经网络系统理论，西安电子科技大学出版社， 1995
[50] 焦李成.神经网络的应用与实现，西安电子科技大学出版社， 1995
[51] 焦李成.神经网络计算，西安电子科技大学出版社， 1995
[52] 庄镇泉，王熙法，王东生.神经网络与神经计算机，科学出版社， 1990
[53] 石纯一，黄昌宁等.人工智能原理，清化大学出版社， 1995
[54] Hunt K. J., Sbarbaro D., Zbikowski R. and Gawthrop P. J., Neural Networks for Control Systems - A Survey, Automatica, Vol. 28, No. 6: 1083 - 1112, 1992
[55] Widrow B., 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation, Proceedings of the IEEE, Vol. 78, No. 9: 27-53, Sept. 1990
[56] Narendra K. S. and Mukhopadhyay S., Adaptive Control of Nonlinear Multivariable Systems Using Neural Networks, Neural Networks, Vol. 7, No. 5: 737-742, 1994
[57] Fuchsia K., A Neural Network for Visual Pattern Recognition, Neural Networks, edited by Clifford Lau, pp. 222-232, IEEE Press, 1992
[58] Lippmann R. P., An Introduction to Computing with Neural Nets, IEEE ASSP MAGAZINE, pp. 5-23, April 198
[59] 吴福朝，张岭.自联想记忆神经元网络的设计，计算机工程， 1996， 22（1）： 23－26
[60] 汪力新，费越，戴汝为.基于人机结合的竞争监督学习，模式识别与人工智能， 1997， 10（3）： 189－195
[61] Rojas R. Neural Networks - A Systematic Introduction, Springer-Verlag Berlin Heidelberg, 1996
[62] 张良杰，李衍达.模糊神经网络技术的新近发展，信息与控制， 1995, 24(1): 39-46
[63] Hecht-Nielsen R., Counter Propagation Networks, Applied Optics, 1987, Vol. 26: 4979-4984.
[64] 张铃，张钹.神经网络中 BP 算法的分析，模式识别与人工智能， 1994， 7（3）： 191－195
[65] 崔大勇，季玫.人工神经网络在自动化中的应用与发展前景，自动化与仪表， 1993， 8（2）： 1－4
[66]杨晓帆，陈廷槐.人工神经网络固有的优点和缺点，计算机科学， 1994， 21（2）： 23－26
[67] 刘伯春.神经网络在控制系统中的应用，电气自动化， 1995， 2： 4－7
[68] 陈荣，徐用懋，兰鸿森.多层前向网络的研究一遗传 BP 算法和结构优化策略，自动化学报， 1997， 23（1）： 43－49